Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert	2019. május 5.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása: 0 % :https://github.com/Davaronas/DE-PTI-Prog1/blob/master/0percent.c 100 % egy szálra : https://github.com/Davaronas/DE-PTI-Prog1/blob/master/100PercentOneThread.c 100 % az összes szálra: https://github.com/Davaronas/DE-PTI-Prog1/blob/master/100PercentAllThreads.c

Tanulságok, tapasztalatok, magyarázat: Több lehetőség is van végtelen ciklust elérni, én a while(1) -et használtam, aztán a 0 %-osnál üresen hagytam ami azt eredményezi hogy nem kerül erőforrásba futtatni , az egy szálasnál a sleep() parancsot használtam, a fork() parancs pedig az összes szálat 100 %-on fogja felhasználni.

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: Ha meg tudnánk mondani melyik program fagy le, akkor alapból nem irnánk olyan programot ami lefagy. A nem lefagyó programból lefagyót készíteni pedig nincs sok értelme, a lefagyóbol pedig nem lehet nem lefagyót készíteni ilyen módon

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: https://github.com/Davaronas/DE-PTI-Prog1/blob/master/VariableSwitch.c

Tanulságok, tapasztalatok, magyarázat: Három egyszerű paranccsal meg lehet oldani a feladatot, segédváltozó bevezetése nélkül, ami a forráskódban látható. Többféle módszer is van rá, de én azt választottam hogy az elsőböl kivonjuk a másodikat, aztán a másodikhoz hozzádjuk a módosított első számot, ezután az a módosított második számból az módosított elsőt, ez lesz a második szám eredménye, és ki is cseréltük őket.

2.4. Labdapattogás

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

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

Megoldás forrása:https://github.com/Davaronas/DE-PTI-Prog1/blob/master/Ball.c

Tanulságok, tapasztalatok, magyarázat: Fontos az ablak meghatározása, mivel ezt használja fel a program hogy irányváltoztasson amit úgy érünk hogy -1-el megszorozzuk a haladás irányát a megfelelő tengelyeken. A for()-t úgy is ki lehet csinálni, hogy while()-t használunk, és amint egyszer elvégzi a feladatot, breakelünk és így ugyanazt érjük el.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása: https://github.com/Davaronas/DE-PTI-Prog1/blob/master/Bogomips.c

Tanulságok, tapasztalatok, magyarázat: A Linux kernel használ ilyet és a program a processzor sebességét méri.

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása: https://github.com/Davaronas/DE-PTI-Prog1/blob/master/PageRank.c

Tanulságok, tapasztalatok, magyarázat: A program a Page-Rank értéket számolja ki, amit a google használ hogy a legjobb értékelésű lapokat hozza be a keresésekre a felhasználó által. Amelyik lapnak jobb az értékelése, annak nagyobb prioriotást ad, és feljebb kerül a sorban. Ezért van az hogy nem véletlenszerűen dobál oldalakat, hanem a hivatalos, eredeti oldalak jönnek ki hamarabb, mint a hamis, vagy nem teljesen a keresett témáról szólóak.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

Megoldás forrása: https://github.com/Davaronas/DE-PTI-Prog1/blob/master/Brun.r

Tanulságok, tapasztalatok, magyarázat: A program az ikerprímekkel foglalkozik, ami még mindig vitatott hogy végtelen van e belőlük vagy sem. Azt jelenti, hogy egy prímszámból kivonjuk az mögötte lévő kisebb prímet, és kettő jön ki. Aztán a két halmazok(az egymásból kivont prímek) reciprokát összeadjuk. Így olyan számokat kapunk amik közelítenek egy számhoz(körülbelül 1,8) de soha nem érik el.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

Megoldás forrása: https://github.com/Davaronas/DE-PTI-Prog1/blob/master/MontyHall.r

Tanulságok, tapasztalatok, magyarázat:Az R instalálása nagyon egyszerű Linuxon, be kell írni consoleba hogy R, aztán leirja hogyan kell telepíteni. A matlabot is instalálltam, ezt követően készen álltam a feladat megoldására. Először nem hoztam létre az stp funkciót, ezért problémába ütköztem, aztán hosszasabb elemzés után rájöttem hogy azért nem tud a FUN egyenlő lenni az stp-vel mivel az nem létezik, azaz nem hoztam létre. Miután ez megtörtént, probléma nélkül csodálhattam meg az elkészített grafikont.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: Itt

Tanulságok, tapasztalatok, magyarázat: Az unáris számrendszer azt jelenti hogy annyi jelet (fontos hogy ugyanaz legyen) teszünk, mint amennyi a decimálisban van. Én ebben a feladatban ' jelet használtam. Egy példa erre, ha beirjuk a programba hogy öt, akkor öt darab ' jelet fog kiirni.

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:Itt

Tanulságok, tapasztalatok, magyarázat: A gyakorlatban a környezetfüggetlen nyelvtan egy makrószerűváltozatát használják. Backus-Naur forma a neve, röviden BNF. A jelölés tömörebb lesz, a nyelve generáló kapacitása nem változik.

```
S, X, Y "változók"
a, b, c "konstansok"
S -> abc, S -> aXbc, Xb -> bX, Xc -> Ybcc, bY -> Yb, aY -> aaX, ↩
    aY -> aa
S-ből indulunk ki

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

```
aabXbcc (Xb -> bX)
aabbXcc (Xc -> Ybcc)
aabbYbccc (bY -> Yb)
aabYbbccc (bY -> Yb)
aaYbbbccc (aY -> aa)
aaabbbccc

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

A, B, C "változók"
a, b, c "konstansok"
A -> aAB, A -> aC, CB ->b Cc, cB -> Bc, C -> bc
S-ből indulunk ki

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

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

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:https://github.com/Davaronas/DE-PTI-Prog1/blob/master/Test.c

Tanulságok, tapasztalatok, magyarázat: A funkciókon belüli váltózó deklarálás a C89-ben nem volt lehetséges, viszont C99-ben igen, a kódcsipet ezt mutatja be egy példán keresztül, de rengeteg más módja is van.

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:https://github.com/Davaronas/DE-PTI-Prog1/blob/master/3.4.l

Tanulságok, tapasztalatok, magyarázat:A program a lexer használatával azt csinálja, hogy a felhasználó által beirt számokat kiirja egészként, utána pedig kiirja a hat tizedesjegyik őket kerekítve.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:https://github.com/Davaronas/DE-PTI-Prog1/blob/master/3.5.l

Tanulságok, tapasztalatok, magyarázat:A lexer segítségével készült ez a program, a felhasználó által írt szöveget "lefordítja" l33t nyelvre. A l33t nyelv betüket, számokat helyettesít olyan betűkkel, számokkal, jelekkel amik (akár több is egy karakterre) hasonlóan néznek ki mint az eredeti karakter.

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)   0-tól 4-ig végigmegy
```

iii.
```
for(i=0; i<5; i++)   0-tól 4-ig végigmegy, de itt post increment-es
```

iv.
```
for(i=0; i<5; tomb[i] = i++)    0-tól 4-ig a tomb elemeihez hozzáad ↩
   egyet
```

v.
```
for(i=0; i<n && (*d++ = *s++); ++i)    0-tól n-ig, a két mutatónak ↩
   ugyanaz az értéke
```

vi.
```
printf("%d %d", f(a, ++a), f(++a, a));   Kiír két decimális számot
```

vii.
```
printf("%d %d", f(a), a);   Szintén kiír két decimális számot
```

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat: A SIGINT azt jelenti hogy 'interrupt from keyboard'. Ctrl-C -vel tudjuk megszüntetni a futó folyamatot a terminálban. A feladatban szereplő sorok nem jól vannak megirva, vagy hiányosak, ettől függetlenül leirtam mellékük mit csinál(ná)nak.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\forall x \exists y ((x<y)\wedge(y \text{ prím})))   Minden x-re létezik ↩
   olyan y ahol x kisebb mint y vagy y prím $

$(\forall x \exists y ((x<y)\wedge(y \text{ prím})\wedge(SSy \text{ prím})) ↩
   )   Minden x-re létezik olyan y hogy x kisebb mint y vagy y prím vagy ↩
   SSy prím (nem tudom mi az az SSy) $

$(\exists y \forall x (x \text{ prím}) \supset (x<y)) Létezik olyan y hogy ↩
   minden x prím és y nagyobb x-nél $

$(\exists y \forall x (y<x) \supset \neg (x \text{ prím})) Létezik olyan y ↩
   hogy minden x nagyobb y-nál és nem 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: A jelentésüket melléirtam a feladatban szereplő sorok mellé. Reménykedem benne hogy jók is.

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; Egész változó
  ```
- ```
  int *b = &a; Egészre mutató mutató referenciája
  ```
- ```
  int &r = a; Referencia "a"-ra
  ```
- ```
  int c[5]; Egészek tömbje
  ```
- ```
  int (&tr)[5] = c; Egészek tömbjének referenciája
  ```
- ```
  int *d[5]; Egészek tömbjére mutató mutató
  ```

- `int *h (); Egészet visszaadó függvényre mutató mutató`
- `int *(*l) (); Egészet visszaadó függvényre mutató mutatójának a mutatója`
- `int (*v (int c)) (int a, int b)`
- `int (*(*z) (int)) (int, int);`

Megoldás videó:

Megoldás forrása: https://github.com/Davaronas/DE-PTI-Prog1/tree/master/3.8.c

Tanulságok, tapasztalatok, magyarázat: A sorok mellé irtam mik azok.

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás videó:

Megoldás forrása:https://github.com/Davaronas/DE-PTI-Prog1/blob/master/4.1.c

Tanulságok, tapasztalatok, magyarázat:Az első sorában az alsó háromszögmátrixnak van egy értéke amit használ, minden egyes ezt követő sorban eggyel több. Fontos hogy megadjuk a mátrix sorainak számát a legelején. A szükséges memória méretet a malloc függvénnyel tudjuk lefoglalni, ezután megkapjuk a mutató mutatóját, ami rájuk mutat. Double **-val tudjuk használni, azért mert ez is egy mutató egy mutatóra. Ezt követően a sorokat megtöltük a megfelelő mennyiségű váltózókkal egy egyszerű ciklus segítségével, hogy megkapjuk az alsó mátrixot.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: https://github.com/Davaronas/DE-PTI-Prog1/blob/master/4.2.c>

Tanulságok, tapasztalatok, magyarázat: Ez a program C nyelvet használva titkosítja a felhasználó által megadott szöveget, az exor logikai műveleten alapszik. A kulcs maximum 100 karaktr lehet. Futtatást így végezzük "./'fájlnév' 56789012 tiszta.txt titkos.szoveg"

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: Itt

Tanulságok, tapasztalatok, magyarázat: Ugyanúgy működik mint a C nyelben megírt EXOR titkosítók, csak Java nyelven. A felhasználó által betáplált szövegből, készít egy titkosított szöveget, egy kulcsot használva.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:https://github.com/Davaronas/DE-PTI-Prog1/blob/master/4.4.c

Tanulságok, tapasztalatok, magyarázat:A C EXOR titkosító által lefordított szöveget dekódolja. Ha kíváncsiak vagyunk mennyi idő alatt dekódolja, használjuk ezt a sort "time ./'fájlnév' titkos.szoveg |grep 56789012"

4.5. Neurális OR, AND és EXOR kapu

R

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

Megoldás forrása: Itt

Tanulságok, tapasztalatok, magyarázat: Az OR, AND és EXOR itt műveleteket jelentenek a bináris számrendszerben. Az Ésnél akkor lesz 1 ha mind a kettő számjegy 1. A Vagynál akkor ha legalább az egyik szám 1, az Exornál pedig ha a két szám különbözik akkor lesz egy tehát 0 és 1, vagy 1 és 0. Több layer-ből áll a program, input, hidden (ebből lehet több is) és az output layer.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:https://github.com/Davaronas/DE-PTI-Prog1/blob/master/4.6.c

Tanulságok, tapasztalatok, magyarázat: Ez egy algoritmus, biológiai idegrendszerek szimulálására való próbálkozás Az emberi agyhoz hasonlóan, csomópontok és irányított kapcsolatok egy összekapcsolt sokaságából van összerakva. Mesterséges neurális hálónak is lehet nevezni. Az emberi agy úgy tanul hogy az ugyanazon impulzus által kiváltott ismétlődő stimuláció hatására változtatja a neuronok közötti szinaptikus kapcsolat erősségét, amit a program is próbál szimulálni.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása: Itt

Tanulságok, tapasztalatok, magyarázat: A program futtatása egy kis odafigyelést igényel először ezt kell beirni: "g++ -c 'fájlnév'.cpp -lpng" ez létrehoz egy .o fájlt, amit ezután ezzel a kóddal kell lefordítani: "g++ -o 'fájlnév' 'fájlnév'.o". Ezt követően pedig futtassuk így: "./fájlnév". Kimenetként egy képet fogunk kapni, amit elment abba a mappába amiben lefutattuk. A program azt csinálja, hogy azokat a számokat, amire a kódban lévő képlet igaz, kiszínezi.

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

Megoldás videó:

Megoldás forrása:Itt

Tanulságok, tapasztalatok, magyarázat: Fordítás: g++ fájlnév.cpp -lpng -O3 -o fájlnév. Futtatásra egy példa: ./fájlnév mandel.png 1920 1080 2040 -0.019473810573093663922605855987058021128 18 -0.019473810572541 0.798505756933826860155534177465597167611 1 0.798505756934379196110285192844457924366. Ugyanaz mint a az előző feladat, csak komplex számokat használ. A fordítást és futtatást hasonlóan végezzük. C-ben minden komplex számnak van egy valós (real) és egy imaginárius (imag) része, amelyet meg kell adnunk mikor deklarálunk egyet.

5.3. Biomorfok

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

Megoldás forrása: Itt

Tanulságok, tapasztalatok, magyarázat:A biomorfokat Clifford Pickover fedezte fel. Itt a g++-t kell használatba vennük a fordításhoz, így: "g++ 'fájlnév'.cpp -lpng -03 'fájlnév'" paranccsal fordítsuk, aztán "./'fájlnév' 'bármi'.png 800 800 10 -2 2 -2 2 .285 0 10" paranccsal futtathatjuk.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:Itt

Tanulságok: Az előző feladatokban használt mandelbrot halmazokhoz képest itt az a különbség hogy az nvdia által fejlesztett CUDA magokat használjuk fel. A CUDA egy párhuzamos számítástechnikai platform és API model, az nvidia által készítve. A CUDA platform egy szofter réteg, ami közvetlen hozzáférést ad a videókártya virtuális utasításkészletéhez. A CUDA platform úgy lett elkészítve, hogy tökéletesen illeszkedjen a C és C++ nyelvekre. Ezt a feladatot a fentiek miatt csakis nvidiás kártyával tudjuk lefuttatni, így: nvcc 'fájlnév'.cu. Futtatás pedig ./'kimenetnév' 'képneve'.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: Itt

Megoldás videó:

Tanulságok, tapasztalatok: Ez is ugyanaz mint az előző feladatok, viszont itt lehet végtelenségig nagyítani a képet, de nem úgy nagyítja hogy csak ráközelít, hanem minden egyes közelítéssel újra is rajzolla a képet. Fordítás: "g++ 'fájlnév'.cpp -lpng -o 'fájlnév'". Futtathatjuk a default értékekkel, így :"./'fájlnév' 'képneve'.png" ekkor a program ki is írja nekünk hogy a default értékeket használja, de viszont a felhasználó is megadhat a programnak adatokat, ezeket hogy hogyan kell megadni szintén a program értesít minket.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása: Itt

Tanulságok, tapasztalatok: Ugyanaz mint a Mandelbrot csak Javaban. Ugyanazokat el lehet mondani mint az előző feladatoknál. Az előző programhoz hasonlóan, a kiszámolt és kirajzolt Mandelbrot halmaz valamely részét fogjuk kinagyítani, aztán onnan is nagyíthatunk egy általunk választott pontjára, és így tovább ameddig szeretnénk. A bal egérgomb lenyomásával, és tartásával kijelölhetünk egy területet, és ezt a területet fogja nekünk a program kinagyítani. A MousePressed() és a MouseReleased() funkciók kerülnek felhasználásra, ezek alapján számolja ki a területet, amire rá kell nagyítani.

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása: Itt

Tanulságok, tapasztalatok, magyarázat: Fontos tudni az eljárásról, hogy egy számítási lépés két normális eloszlású számot állít elő, tehát minden páratlanadik meghíváskor nem szükséges számolnunk, csak az előző lépés másik számát visszaadnunk. Érdekesség, hogy a Java development kit Random.java fájlában majdnem ugyanúgy van leírva ez az algoritmus. g++ 'fájlnév'.cpp -o 'fáljnév' parancs kell a fordításhoz és ./'fájlnév' parancs kell a futtatás.

6.2. LZW

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

Megoldás videó:

Megoldás forrása: Itt

Tanulságok: Az LZW (Lempel-Ziv-Welch) egy veszteségmentes tömörítési algoritmus, amely a program egy felhasználó által beadott szöveges fájlból készít egy nullákból és egyesekből álló fát. Terry Welch publikálta 1984-ben ezt az algoritmust. g++ 'fájlnév'.cpp -o 'fáljnév' a fordítás és ./'fájlnév' 'textfájlnév' a futtatás.

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: Itt

Tanulságok: Itt a lényeg hogy sorrendben járjuk be a fát. Kiírási iránya lehet postorder és preorder is, attól függően melyik programot futtatjuk. Preorder bejárás azt jelenti hogy a gyökér elem majd a bal oldali részfa preorder bejárása, végül a jobboldali részfa preorder bejárása. Postorder pedig úgy végezzük hogy először a bal részfa posztordert bejárjuk, majd a jobboldali részfa posztordert, végül feldolgozzuk a gyökérelemet. g++ 'fájlnév'.cpp -o 'fáljnév' a fordításhoz és ./'fájlnév' a futtatáshoz

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: Itt

Tanulságok:Itt a gyökér az objektum, a fa pedig mutató. Arra a csomópontra mutat a fa amelyik csomóponton járunk. g++ 'fájlnév'.cpp -o 'fáljnév'-vel fordítjuk, és ./'fájlnév'-vel futtatjuk.

6.5. Mutató a gyökér

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

Így kell átírni:

```
        class LZWBinfa
{
public:
  LZWBinfa ()
  {
    gyoker = new Csomopont();
    fa = gyoker;
  }
  ~LZWBinfa ()
  {
    szabadit(gyoker->egyesGyermek ());
    szabadit(gyoker->nullasGyermek ());
    delete gyoker;
  }


        A gyökerek elől ki kell törölni a referencia jeleket
```

Tanulságok: Annyi a változtatás hogy a fa gyökere is objektum, a fával együtt mostmár ez is mutató lesz. Fordítás: "g++ 'fáljnév'.cpp -o 'fájlnév'". Futtatás: "./'fájlnév'"

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása: Itt

Tanulságok: Itt az változik hogy van a programban egy mozgató konstruktor. A felesleges másolások elkerülésére használatos. A mozgatás olcsóbb művelet mint a másolás. Fordításhoz szükség van egy BT.h nevű fájlra, amit meg lehet találni a github repomban. Fordítás: "g++ 'fáljnév'.cpp -o 'fájlnév'". Futtatás: "./'fájlnév'"

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: Itt

Tanulságok, tapasztalatok, magyarázat: A program a hangyák viselkedését próbálja meg utánozni. A képernyő cellákra bontódik, ezután a hangyák megkeresik azt az egyedet, akinek a legerősebb a szaga és elindulnak abba az irányba. Ennek a programnak a fordításához szükséges a Qt. Ennek a programnak meglehetősen testreszabhatóak a tulajdonságai. Ezeket megtalálhatjuk a main.cpp-ben leírva. W kapcsoló a szélességet befolyásolhatja, m kapcsolóval meg tudjuk adni a magasságot, az n kapcsoló arra szolgál hogy megadjuk a hangyák számát, a t a sebességüket, p a párolgás mértékét, f a feromonszintet, másképp a hagyott nyom értéke, s a hagyott nyom értéke a szomszédokban, d az alapérték vagyis az induló érték a cellákban, a kapcsolóval a maximum mennyiségű cellákat tudjuk befolyásolni, i-vel pedig a minimumot, a c kapcsolóval pedig a cellaméretet tudjuk megadni.

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása: https://github.com/Davaronas/DE-PTI-Prog1/blob/master/7.2.java

Tanulságok, tapasztalatok, magyarázat: Ugyanaz az elven működik mint a C++ életjáték, csak Javában. John Horton Conway által behozott szabályok ugyanazok mint a C++ verzióban.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: Itt

Tanulságok, tapasztalatok, magyarázat: John Horton Conway konstruálta meg a neumanni értelemben vett önreprodukáló gépeket 1970-ben. Életjáték a neve. A 2D-s tér nem végtelen, hanem periodikus határfeltétellel működik, azaz ami fent elhagyja a képernyőt az alul vissza fog jönni. A játékostól nem kér inputot, magától történik a játék. Egy sejt meghal, ha kevesebb mint kettő élő szomszéddal rendelkezik. Háromféle entitás van a játékban, "Oscillator", "Still life" és "Spaceship". Bizonyos lépésszám után az "Oscillator" visszatér kezdeti alakjához. A "Still life"-ok nem csinálnak semmit se, a "Spaceship"-ek pedig tudnak mozogni.

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: Itt

Tanulságok, tapasztalatok, magyarázat: Ez a program egy teszt ami a fókuszáló képességünket méri le, mennyire vagyunk képesek zavaró tényezők mellett is koncentrálni, ami azt szolgálja hogy mennyire lennék jó esportolók. A játékosnak követnie kell a saját karakterét és közben pedig figyel olyan dolgokat hogy hányszor vesztjük szem elől a karakterünket és meddig tart újra megtalálni. Ezt követően a játékos kap egy összesített pontszámot, ami értékeli a teljesítményét. Nincs bebizonyítva viszont hogy ez tényleg leméri e mennyire lennék jó esportban.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Megoldás videó:

Megoldás forrása: https://github.com/Davaronas/DE-PTI-Prog1/blob/master/szoftmax.py

Tanulságok, tapasztalatok, magyarázat: Ehhez a programhoz a tenserflowra lesz szükségünk, mivel az MNIST egy tenserflow alapú könyvtár. Az a lényege hogy mesterséges inteligenciákat (Artifical intelligence) lehet rajtuk tesztelni, és "edzeni". A program nem feltétlenül teljesen pontos, de viszont minél nagyobb mennyiségű mintát adunk meg neki, annál jobban képes lesz pontosabb megoldást biztosítani. Ebben a feladatban számokat tanítunk meg "kiolvasni" a programnak. Érdekes megjegyezni hogy hasonló módszert használnak arra is hogy valakinek felismerjék az aláírását elektornikusan. Itt is a pontosság jobb lesz ha több aláírást adunk be a programnak.

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:https://github.com/Davaronas/DE-PTI-Prog1/blob/master/deep_mnist.py

Tanulságok, tapasztalatok, magyarázat: Ugyanaz mint az előző program, viszont itt komplexebb számokkal is képes dolgozni, mivel több rétegen is átmennek a számok, és ezek után miután az összes kielemezte meghozza meg a döntését. El lehet mondani hogy emiatt pontosabb is a megítélés. Természetesen ehhez a programhoz is szükséges a tenserflow telepítése mivel az adatbázisából dolgozik.

8.3. Minecraft Malmö

Megoldás videó:

Megoldás forrása:https://github.com/Microsoft/malmo

Tanulságok, tapasztalatok, magyarázat:A Minecraft Malmo egy fejlett mesterséges intelligencia fejlesztésére a próbálkozás. Tesztelik hogy egy komplex környezetben hogyan lehet AI-t csinálni, Minecraftra építve. Hatalmas variáció van amire meg lehet tanítani az AI-t Minecraftban mivel akár mehetünk végtelenségig egyenen, felfedezhetjük a környezetünket, végigmehetünk egy dungeonon, vagy építkezünk, akár hatalmas monumentumok felépítésére is megtaníthatjuk a karaktert. Nyitott forráskódú tehát bárki aki szeretne ezzel foglalkozni megteheti. A Microsoft indította néhány évvel ezelőtt. Sok programozási nyelveket támogat, például java, C#, C++.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása: https://github.com/Davaronas/DE-PTI-Prog1/blob/master/9.1.lisp

Tanulságok, magyarázat: Az első egy rekurzív függvény, amely önmagát hívja addig, amíg nem lesz nulla. A második függvény is önmagát hívja, és szorzásokat végez amíg el nem ér egy bizonyos számot. A programban nincsenek olyan sorok amik kiiratnák ezeket a számokat, valószínűleg úgy volt tervezve hogy ezek a sorok funkciót fognak alkozni, és egy másik kódból meg lesznek hívva. A Lisp nyelvet matematikai feladatokhoz, és mesterséges intelligencia fejlesztéshez használták először.

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://github.com/Davaronas/DE-PTI-Prog1/blob/master/9.2.scm

Részletes leírása a lépéseknek: http://penguinpetes.com/b2evo/index.php?p=351

Tanulságok, tapasztalatok, magyarázat: Ez a program a scheme nyelvvel van megírva, eléggé hasonlít a lisphez. Script-Fu-val lehet írni a gimphez kódokat. Ez abban segíthet minket, például ha többször el akarjuk ugyanazt a műveletet végezni végezni, akkor ezt megírjuk és a programunk elvégzni helyettünk. Abban is segít ha olyan műveletet akarunk végezni amire a gimp nem képes.Ez a program a beadott szövegre króm (fémes) hatást tesz. Ehhez a program kiszámolja a pixeleket amiket át kell alakítani, aztán kiszámolja hogy hogyan kell átalakítani, és végül elvégzi a módosításokat.

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://github.com/Davaronas/DE-PTI-Prog1/blob/master/9.3.scm

Tanulságok, tapasztalatok, magyarázat: Ez a program a beadott névből készít egy mandalát. Leírjuk a szót amit szeretnénk mandalává alakítani, aztán egy új rétegre leírjuk megint, és 180 fokkal elfordítjuk. Újra meg újra ilyen forgatásokat végzünk, addig amig ki nem jön egy kör szerűség. Ezt az egészet utána egy körrel körbe vehetjük. Színeket beállítjuk tetszés szerint, és készen is van a mandalánk a szóbol amit beirtunk, a nevünkkel szokás ezt csinálni.

10. fejezet

Helló, Guttenberg!

10.1. Guttenberg olvasónapló

KERNIGHANRICHIE

Azt a kifejezést ami pontosvesszővel le van zárva (utasításlezáró jel) utasításnak nevezünk. Ezt követően a kacsos zárójelekkel közrefogva tehetjük ide a parancsainkat, deklarációinkat. A zárójel végét nem követi pontosvessző.

If else

Akkor használjuk ha valamilyen feltétel bekövetkezésekor akarunk valamit hogy megtörténjen, az 'else' nem feltétlenül szükséges. Ha csak egy parancsot akarunk ez után elvégezni nem szükséges kapcsos zárójelet használni, de amint többet akarunk szükséges. A jobb átláthatóságért egy parancsnál is hasznos lehet. Ezeket egymásba lehet ágyazni és végtelenségig lehet 'if else'-t használni. Viszont ha egy pár darabnál többet akarunk ajánlatos a következő parancs használatát megfontolni.

Switch

Ha több irányba el akarjuk ágaztatni a programunkat nagyon hasznos parancs. Itt is megadott feltételeknek kell teljesülni hogy egy irányba menjen. Ha nem felel meg egy kritériumnak sem, akkor a default úton fog menni a program, ha raktunk bele default utat. Az egyes elágazások után break-et kell használni.

While, For

Ciklusoknak nevezik őket, a bennük lévő kifejezés elemzése után megtörténnek a benne lévő parancsok, akkor ha igaz a kifejezés. Amint ez megtörtént a kifejezést újra elemzi és addig újra és újra elemzi és végrehajtja a parancsokat ameddig a kifejezés nem lesz hamis.

Do-while

Egyszer mindig végre fog hajtódni, csak ezután nézi meg a kifejezés igazságértékét, és ha igaz megint végrehajtja. Addig ezt teszi amig a kifejezés nem lesz hamis

Break, Continue

A break garantálja azt, hogy ki tudjunk lépni egy ciklusból, mielőtt újra kiértékelésre kerülne. A continue pedig a következő iterációs lépést teszi meg.

Goto

Nem egy gyakran használt parancs, különböző cimkékre lehet vele ugrálni, akár eljárás vagy függvényből is ki lehet ugrani.

BMECPP

C++ nyelv használatánál függvényeknél ha nem akarunk paramétert megadni, akkor használni kell a void szót, ha üresen hagyjuk C-ben ez is működőképes, viszont C++-ban nem. Valamint visszatérési értéket kötelező megadni C-vel ellentétben.

A main függvényben használhatjuk az argc változót és az argv-t, ezután return 0-val folytathatjuk, de nem muszáj hiszen alapértelmezettként 0-át ad vissza ha lefutott a programunk sikeresen.

A bool típus lehet 'true' vagy 'false', rengeteg féleképpen lehet használni, feltétel teljesítése érdekében használják a legtöbbet.

Több bájtos stringek, beépített típus, nem szükséges include-olni.

Változót deklarálás állhat bárhol ahol egyébként utasítás is. Dönthetünk úgy is hogy egyből adunk kezdőértéket, vagy később.

C++-ban lehetőségünk van alapértelmezett értékeket használni, ha nem adtunk meg paramétert a függvénynek. Az argumentumlistában hátulról előre haladva adunk meg alapértelmezett értéket. Híváskor ugyanezt a sorrendet követi.

Függvényeknél C-ben érték szerinti paramerátadást történik, C++-ban pedig hivatkozás szerint van megoldva. Ha módosítani is szeretnénk a paramétert akkor alkalmazzuk ezt a módszert. A kiszemelt paraméterhez hozzáteszünk egy and jelet, így tudatjuk, hogy a paraméternek a referenciáját akarjuk használni.

PICI

Az első fejezetet a 16. oldaltól érdemes kezdeni olvasni, az előző oldalak a FORTRAN-ról és COBOL-ról szól. Ez a fejezet programozási alapfogalmakról szól, a programozásnak a céljáról. Megemlít különböző nyelveket, imperatív és deklaratív nyelveket.

A második fejezet a programozási nyelvek alapeszközeiről, alapfogalmairól szól, mint például karakterkészlet, kulcsszavak, lexikális egységek. Általános szabályokat is megismerünk a programozásról. Megtalálhatunk benne különböző nyelvekben példákat hogy ezeket bemutassa. Az adattípusokról lesz szó ezt követően, változókról és nevesített konstansokról.

Kifejezésekkel ismerkedhetünk meg a következő, harmadik fejezetben. Operandus, operátor, prefix, inflix, postflix alak. Megemlíti a típuskényszerítési művelet is.

A negyedik fejezetben utasításokról van szó, deklaratív és végrehajtó utasítások. Értékadó utasítás, ugró utasítás(goto), elágazó utasítás(if-else,switch), mint példákkal bemutatva.

Az ötödik fejezet a programok szerkezetével, felépítésével foglalkozik. Szó kerül a paraméter adásról. A zárójelek a hatáskört kezdetét és végét jelzik.

A hatodik fejezetben az absztrakt adattípusokról tudhatunk meg információkat. Ez egy olyan adattípus ami amely megvalósítja a bezárást és az információ elrejtést. Az objektumorientáltságra gondolhatunk főleg. Meghatározott és szigorú műveletekkel juthatunk hozzá az értékekhez.

A hetedik és nyolcadik fejezet főleg az Adáról szól.

A kilencedik fejezet pedig a kivételkezelésről szól. Ezzel megtehetjük hogy elvesszük az operációs rendszertől a hibakezelés folyamatát.

Githubon

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.