Univerzális programozás

Így neveld a programozódat!

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

KÖZREMÜKÖDTEK			
	CÍM : Univerzális programozás		
HOZZÁJÁRULÁS	*NÉV*	*DÁTUM*	*ALÁÍRÁS*
ÍRTA	Bátfai Norbert és Őz Ágoston	2019. március 12.	

VERZIÓTÖRTÉNET			
VERZIÓ	DÁTUM	LEÍRÁS	NÉV
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:

- Egy mag 0%.
- Egy mag 100%.
- Összes mag 100%.

A számítógépen egy végtelen ciklus futtatása közben megközelítőleg 0%-os CPU terhelést a "sleep (1)" rendszerhívással tudunk elérni. Ez esetben a következő 1 milliszekundumban nem fogunk CPU-időt kapni így nem lesz leterhelve a CPU. Egy mag 100%-os felhasználására elég csak egy végtelen ciklust írnunk. Az összes mag/szál (számítógépem esetében két mag és annak négy logikai szálja) 100%-os kihasználására az egyik lehetőség "forkolni", mely által egy adott szülőprocesszsnek csinálunk gyermekprocesszeket (akár többet is), melyek egymás mellett futnak ugyanarra az utasításra. Esetemben négy végtelen ciklus, melyekből hármat "forkoltam" elegendőnek bizonyultak kihasználni a CPU összes magját/szálát 100%-on.

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

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

}
```

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

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

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

A programozás és a lefagyó program problémája szinte egyidősnek számítanak. Alan Turing (akinek a neve köthető a Turing-gép fogalmához) bizonyította be matematikailag, hogy efféle programot lehetetlenség írni. Egy könyebb, kevesebb sorból álló programnál el lehet dönteni, hogy le fog fagyni, de egy bonyolult, hosszú sorokból álló és összetett programnál ezt képtelenség megmondani.

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:

- Változók értékének cseréje.

Két változó értékét az egyik legegyszerűbben úgy tudjuk felcserélni, hogy beiktatunk egy harmadik változót a programba. Így, ha az értékeket egyenlővé tesszük egy másik választottal. Pl: a, b, c esetében c legyen egyenlő a-val, a legyen egyenlő b-vel és b legyen egyenlő c-vel. Ezt sorrendben elvégezve az értékek felcserélődnek. Azonban logikai utasítás vagy kifejezés nélkül csak a két megadott változóból indulhatunk ki. Ebben a helyzetben pl. pár logikus összeadás és kivonás elvégzése után az értékek szintén felcserélődnek.

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:

- Labdapattogtatás.
- Labdapattogtatás if nélkül.

Ebben a programban az első lépés a megfelelő függvény megtalálása volt egy "labda" (igazából egy karakter, mint pl: "O") "pattogtatására" (terminál ablakban való megjelenítésére). Lényeges az is, hogy az ablakunkból ne menjen ki a "labda". Az "ncurses" függvénykönyvtár alkalmas használatra, mivel terminálban dolgozunk. A változók értékeit igény szerint beállítjuk mindkét tengelyen, majd az "mvprintw" függvénnyel kiiratjuk pl. az "O" karaktert. A "refresh" függvénnyel tudunk kimenetet láttatni a terminálban, a "usleep"-el pedig be tudjuk állítani milyen időközönként is rajzoljon ki egy karaktert a program.

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:

- Szóhossz.

Egy szó hosszának megszámolására elsőnek egy "int" változót kell létrehoznunk a programban és ezt egyenlővé kell tennünk eggyel. Balra léptetve, ameddig csak lehet és számolva a léptetések számát végül megkapjuk eredményül mennyire hosszú az adott szó amit vizsgálunk.

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:

- PageRank.

A PageRank algoritmust a Google-nél fejlesztette ki Larry Page és Sergey Brin (Page-ről kapta meg a nevét). Gondolatmenetük alapján minél több oldal mutat rá egy adott oldalra, annak annál nagyobb az értéke. Gyakorlatban persze ez nem ilyen egyszerű, viszont egy oldal fontossága valóban látható azáltal, hogy hány másik oldal is mutat rá. Természeten a Google a PageRank algoritmus egy teljesen átdolgozott változatát használja (az is lehet, hogy az merőben eltér az eredetitől). Annak a forráskódját nem is publikálták, ami egyértelmű, hiszen máskülönben egy kis munkával és utánajárással akárki indíthatna keresőszolgáltatást.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

A prímszámok érdekessége az, hogy csak eggyel és önmagukkal osztva nem adnak maradékot. Az olyan prímeket melyeket kivonva egymásból kettőt kapunk maradékul, ikerprímeknek nevezzük. A Brun-tétel azt mondja ki, ha az ikerprímek reciprokát vesszük és elkezdjük őket összeadni, akkor azok egy konstanshoz fognak konvergálni. Ez a tétel viszont önmagában nem megoldás az ikerprímek számának problémájára, ugyanis azt nem mondja ki, hogy az összeadásokból kapott hosszú sor véges-e vagy végtelen.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Az R programozási nyelv főleg statisztikai számításokra alkalmas, interpreteres nyelv. Ez az egyik oka annak, hogy könnyen lehet olvasni és értelmezni a Monthy Hall problémára megalkotott R program forrását. Az említett probléma egy TV show-ból származtatható, ahol három ajtó áll előttünk és ezek közül kell választanunk. Az egyik ajtó kincset rejt, míg a másik kettő mögött nem található semmi. A szimuláció által azt vizsgáljuk, hogy melyik az érdemesebb: ugyanannál az ajtónál maradni a játék körei alatt vagy váltogatni az ajtókat végig. A program kimenetét vizsgálva azt tudjuk megállapítani, hogy 100000000 lefutás után az eredmény az, hogy igenis érdemes váltogatni az ajtókat a játék alatt, így akár 50%-os javulást is elérhetünk.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Megoldás forrása:

- Decimálisból unárisba.

Az unáris számrendszer vonalakkal vagy húzásokkal ábrázol számokat. Egy vonal eggyel egyenlő és ezeket összeadva lehet értelmezni. A Turing-gép ezt az átváltást végzi.

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

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

I. Változók: S, X, Y. Konstansok: a, b, c. Szabályok: S->abc, S->aXScc, X->aYa, Ya->aX, Xa->aabb. Így: S->aXScc->aXabccc->aaabbbccc.

II. Változók: S, X, Y. Konstansok: a, b, c. Szabályok: S->abc, S->aXaYS, Yab->bcc, Xa->aabb. Így: S->aXaYS->aXaYabc->aXabccc->aaabbbccc.

A formális nyelvek és nyelvtanok legjelentősebb úttörője Noam Chomsky. Munkássága egyaránt hatott a formális- és a természetes nyelvek kutatására is. A generatív nyelv a legismertebb, ez azoknak a szabályoknak a halmaza, amelyekkel a nyelv minden lehetséges jelsorozata előállítható, avagy hogyan is vagyunk képesek átírási eljárásokkal előállítani a kezdő szikbólumokból a többi jelsorozatot a szabályok szerint.

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:

- C89 és C99.

Egy adott nyelv vezérlésátadó utasításai az egyes műveletek végrehajtási sorrendjét határozzák meg. A C programnyelvben több utasítás fajta is létezik. Ilyen a kifejezés utasítás, összetett utasítás, iterációs utasítás, vezérlésátadó utasítás, kiválasztó utasítás és a címkézett utasítás. Egy kifejezés utasítássá válik, ha egy pontosvesszőt írunk utána. Kapcsos zárójelekkel utasítások csoportját tudjuk összefogni egyetlen utasítás blokkba. Az "if" és "else" utasításokat döntés kifejezésére használjuk, itt az "else" rész opcionális. Az "else if" utasítás adja a többszörös döntések általános szerkezetét. A "switch" utasítást is hasonlóan alkalmazzák. A "while for" utasításnál a program először kiértékeli a kifejezést, ha nem nulla az érték végrehajtja az utasítást és a kifejezés újra kiértékelődik, ez a ciklus akkor ér véget, ha a kifejezés egyenlő lesz nullával. A "break" utasítás a ciklusokból való idő előtti kilépést teszi lehetővé (legbelső ciklusból lép ki mindig). A "continue" utasítás a "break" utasításhoz kapcsolódik, ez megkezdi a következő iterációs lépést. A "goto" utasítással egy megadott címkére ugorhatunk.

3.4. Saját lexikális elemző

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

Megoldás forrása:

- Lexikális elemző.

Előrször meghívjuk a függvénykönyvtárat. Ezután megadunk egy számlálót a lexernek. A szabályokat a "%% %%"-ban tudjuk leírni. A "[[:digit:]]+ megad több számot is egymás után, majd növeljük az értéket eggyel. A "[a-zA-Z] [a-zA-Z0-9]*" az összes alfanumerikus karakter láncért felelős. A main-ben az "yylex" segítségével meghívjuk a lexert, ez bájtonként fog haladni és végig fog futni a bemeneten. Végül csak kiíratjuk az eredményt.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás forrása:

- l33t cipher.

Elsőnek meghívjuk a függvénykönyvtárakat. Létre kell hoznunk egy int típusú változót. A program működéséhez nélkülözhetetlen egy cipher típusú tömb létrehozása, ez a betűkhöz és számokhoz tartozó lehetséges leet kódokat tartalmazza (általában négy változatlan és három kódolt betűt). A lexer által beolvasott karakterekre megnézi a program, hogy van-e köztük olyan, amely benne van a cipher típusú tömb kódolandó karakterei között. Ha igen, akkor véletlenszerűen kiválaszt egy kódolást és kiírja. Ha nem, akkor pedig a változatlan karakter fogja kiírni.

3.6. A források olvasása

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

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

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

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

---

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

Ha a SIGINT jelkezelés nem lesz ignorálva, akkor innentől kezdve végezze a jelkezelést.

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

Ötször végezzük el ...

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

Ötször végezzük el ...

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

Nem lehet eldönteni, mi is fog történni, mert az i-t már használtuk előzőleg. Nem érdemes ilyen kódot írni.

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

Nehezen olvasható a kód és nem egyértelmű a kiérékelési sorrend. Itt i legyen egyenlő nullával és ha i kisebb mint n, valamint d rákövetkezője megegyezik s rákövetkezőjével, akkor növeljük i-t eggyel.

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

Egy "printf" függvényben kiértékelni változókat nem jó megoldás, hiszen nem lehetünk biztosak, hogy milyen sorrendben fog ez megtörténni. Nagy valószínűséggel nem kívánt eredményt fogunk kapni.

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

Kiíratjuk az f függvényben szereplő a-t és magát a-t.

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

Az f függvény referencia szerint fogja megkapni az a változót és így módosítani tudja azt. Így nem lehetünk biztosak a kiértékelési sorrendben.

Megoldás forrása:

- Jelkezelő.

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

Helyesen kiolvasva a fenti kifejezéseket ezeket kapjuk.

I. Minden x esetén létezik olyan y, ami nagyobb mint x és y prím. Ebből az következik, hogy végtelen sok prímszám létezik.

II. Minden x esetén létezik olyan y, ami nagyobb mint x és y prím, valamint ikerprím. Így végtelen sok ikerprím szám létezik.

III. Létezik olyan y, ami minden x esetén nagyobb ha x prímszám. A supset az "implikáció" a nyelvben. Ez azt jelenti, hogy véges sok prímszám létezik.

IV. Létezik olyan y, ami minden x esetén igaz, hogy ha y kisebb mint x, akkor x nem prímszám. A "negáció"-val tagadom, hogy x prím. Így véges sok prímszám létezik.

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

  Egészre mutató mutató.
- ```
  int &r = a;
  ```

  Egész referenciája C++ -ban.
- ```
  int c[5];
  ```

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

  Egészek tömbjének referenciája.
- ```
  int *d[5];
  ```

  Egészek tömbjére mutató mutató.
- ```
  int *h ();
  ```

  Egészre mutató mutatót visszaadó függvény.
- ```
  int *(*l) ();
  ```

  Egészre mutató mutatót visszaadó függvényre mutató mutató.

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

  Egészet visszaadó és két egészet kapó függvényre mutató mutató visszaadó, egészet kapó függvény.
- ```
  int (*(*z) (int)) (int, int);
  ```

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

Megoldás forrása:

- Deklaráció.

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás videó:
Megoldás forrása:
Tanulságok, tapasztalatok, magyarázat...

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!
Megoldás videó:
Megoldás forrása:
Tanulságok, tapasztalatok, magyarázat...

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!
Megoldás videó:
Megoldás forrása:
Tanulságok, tapasztalatok, magyarázat...

4.4. C EXOR törő

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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...

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

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

Megoldás videó:

Megoldás forrása:

5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

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 videó:

Megoldás forrása:

6.3. Fabejárás

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

Megoldás videó:

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:

Megoldás forrása:

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

9.3. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.4. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. Omega

Megoldás videó:

Megoldás forrása:

III. rész

Második felvonás

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

10. fejezet

Helló, Arroway!

10.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

10.3. Általános

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

10.4. C

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

10.5. C++

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

10.6. 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.