Univerzális programozás

Így neveld a programozód!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert Ã©s Halasz, Adam	2019. május 16.	

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

Ajánlás

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

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

Tartalomjegyzék

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

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

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: https://github.com/Drcsonka/Prog1/blob/master/infinite_loop.c

A programba 3 feladat rész van. Az első feladatrészben 1 CPU mag 100%-on fut ezt egy vegtelen ciklussal erjuk el, a masodikban 0 CPU fut 100%-ba ezt a sleep parancsal kapjuk meg, és a harmadikban az összes 100%-ban fut itt kapcsolok hasznalataval oldjuk meg, hogy a vegtelen ciklus mindegyik CPU-t egyszerre hasznalja.

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 program sosem mukodik rendesen, mivel ha az ha mindketto program true-t ad vissza akkor van benne vegtelen ciklus, ha viszont az elso programban nincs vegtelen ciklus akkor a masodik program fog vegtelen ciklusban maradni.

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/Drcsonka/Prog1/blob/master/valtozocsere.c

A minusz előjel felhasználásával, és a +, - matematikai műveletekkel, sikeresen meg tudjuk cserélni a két változó értékét. Ez a legegyszerűbb módja a változók felcserélésére.

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/Drcsonka/Prog1/blob/master/Labdapattog.c https://github.com/Drcsonka/Pr

Az 'x' es az 'y' tengelyt növeljük, ezzel a képernyőn lévő labda mozog, majd amint eléri az ablak egyik falát akkor az annak megfelelő tengelyt ellentétesen megváltoztatjuk, hogy a masik iranyba mozogjon. Ezzel olyan hatást keltve mintha lepattant volna a falról es a másik irányba kezdett volna mozogni.

A program elején deklaráljuk a labda koordinátáit. Az 'if'-es megoldásban ez az 'x' és 'y' változók lesznek, miközben az 'if' nélküli megoldásban 'xj', 'xk' és 'yj', 'yk'-k lesznek.

A program a 'clear()' function letisztítja számunkra a képernyőt. A 'refresh()' function újratölti a képernyőt.

Az 'if' nélküli feladatot maradékos osztással oldjuk meg.

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/Drcsonka/Prog1/blob/master/shifteles.c https://github.com/Drcsonka/Prog1/blob/master/shifteles.c

Felveszünk egy változót, és addig shifteljük, ameddig nem lesz az értéke 0, vagyis ameddig minden érték le csúszik. A shiftelések száma lesz a szónak a hossza. Egy int változó 4 bit-et foglal így annyiszor 4 bites lesz a szavunk amennyi betűs.

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/Drcsonka/bevprog/blob/master/ora4v2.cpp

Tanulságok, tapasztalatok, magyarázat...

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

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat...

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/unaris.cpp

Az Unáris vagyis masneven Egyes szamrendszer, egy olyan szamrendszer amiben egy szamot ugy abrazolunk, hogy annyiszor egymassmelle tesszuk pl.: .... = 4 Főleg kis értékek / számok esetén működik jól. Megadunk egy számot a programnak és egy 'for' ciklus segítségével annyiszor egymás mellé írjuk az a karaktert amit megadtunk (pl : .) amennyi a beírt szám értéke

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: https://github.com/Drcsonka/Prog1/blob/master/kornyezetfuggetlen.txt

Noam Chomsky nevéhez fűződik a generatív grammatika nyelv. A Grammatikanak 4 alkotóeleme van a : A nemterminális jelek vagy változó szimbólumok, általában nagybetűkkel (A, B, C) jelöljük . A terminális jelek, vagy konstansok, általában kisbetűkkel (a, b, c) jelöljük. A helyettesítési szabályok, elemeit rendezett párok képzik. A kitüntetett nemterminális jel, a grammatika kiinduló vagy kezdő eleme.

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/Drcsonka/Prog1/blob/master/c89.c https://github.com/Drcsonka/Prog1/blob

A programban egy ciklus található ami 1-10ig megy el.C99-ben le fog futni miközben, c89-ben nem fog, mivel ebben a verzióban még nem lehet változót deklarálni a ciklus fejben.

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/Drcsonka/Prog1/blob/master/Lexikalis.c https://github.com/Drcsonka/Prog

Tanulságok, tapasztalatok, magyarázat...

A lex program segítségével, nekünk csak szabályokat kell megadnunk, ami alapján használható szabványos programkódot fog előállítani.

A programunk 3 részből áll.

Az első rész C kódot tartalmaz.

A második részben szabályokat definiálunk. A reguláris kifejezésekkel kezdünk. A másik a program által kapott szöveget fogja elemezni. Az 'atof()' függvénnyel fogjuk a 'string'-eket double típúsra átalakítani.

A harmadik részben található a 'yylex()' függvény ami a második részben megadott szabályokat hívja meg. Majd kiírajta az eredményt.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/leet.l

A l33t nyelv lényege, hogy a betűket rájuk hasonlító karakterekkel és számokkal helyetessíti. Ez a program is a lex program használatával fog működni.

Az program első részében a 'L33SIZE' elemet fogjuk definálni. Ez fogja a majd helyettesíteni az kért szöveget. Meg van adva egy 'c' változó amiben az alap karakter van és a '*leet[4]' tömben a 4 változata van amire kicserélhetjük. Ebből a 2 változóból fog összeállni a 'l337d1c7[]' tömbünk.

Majd megvizsgáljuk egyesével a karaktereket egy reguláris kifejezéssel. Meghívjuk egy 'for' ciklusban a 'L33SIZE' konstanst, ide fog majd behelyettesíteni.

A programunk nem tud mit csinálni az uppercase karakterekkel ezért van a 'tolower' függvényünk ami átfogja ezeket allakítani kisbetűre. Majd generálunk egy random számot 1-100ig, és ez alapján döntjük el, hogy a 4 változat alapján melyikkel helyettesítsük karakterünket.

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);
```

Ez a kód részlet azt jelenti, hogy ha a 'SIGINT' jel kezelése nem volt figyelmen kívül hagyva akkor ezután se legyen.

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

Egy 'for' ciklus fog lefutni ami 0-4ig fogja kiírni a számokat egyesével. Mivel a '++i' változót használata előtt növelem, így 5-re növelve már nem fut le mégegyszer, hanem vége lesz a ciklusnak.

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

Ez a ciklus hasonlít az előzőhöz, annyi különbség van a kettő között, hogy itt az 'i' változót először használjuk és utánna változtatjuk.

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

A 'for' ciklus mind eddig is 0tól addig megy még el nem éri az 5-öt. Közbe a 'tomb' i-edik elemébe belerakja az 'i' értékét is. A program le fog futni de, bugos, és nem a vár eredményt kapjuk. 'Splint' lefutattása szerint is hibás.

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

A ciklusunknak addig kéne futnia ameddig 'i' el nem éri az 'n' értékét, de emelett van egy másik feltételünk is, hogy a 'd' és az 's' pointer 'egyenlő' nem lesz, miközben növeljük őket. De ez a rész csak egy '='-et tartalmaz ami magában nem megfelelő, így bugos a program.

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

A programkódnak kiíratja az 'f()' függvény értékét kétszer. A probléma ott lesz, hogy az első meghívásnál megváltoztatjuk 'a'-nak az értékét amit majd a másodikban felhasználunk újra. Mivel a kiírtékelési sorrend nincs előre megadva így bugos lesz.

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

Ez a programkód már helyes. Az 'f()' függvény egy paramétert kér amit az 'a'-val megkap, és ennek az eredménye lesz kiírva.

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

Itt is az 'f()' függvény értékét fogjuk kiírni, csak azzal a különbséggel, hogy egy memóriacímre mutató értéket fog kapni a függvény.

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/Forrasolv.c

Megoldás videó:

A program lefutasa elohiv egy vegtelen ciklust, amibol a CTRL + C billentyukombinacioval sem tudunk kilepni.

A 'splint' egy olyan eszköz ami segít megnézni 'C'-ben írt kódokat, és azokban hibákat találni.

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

A feladat megoldásához a LaTeX szövegformázó rendszert kell használnunk. A feladat leírásában olvasható parancsokat a szoftver matematikai kifejezésekké alakítja át, majd PDF formában lementve mindenki számára jól értelmezhető formát kapnak.

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:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/Deklaracio.c

A fajlban fel vannak tuntetve, hogy melyik a milyen valtozot deklaral a program azon resze.

4. fejezet

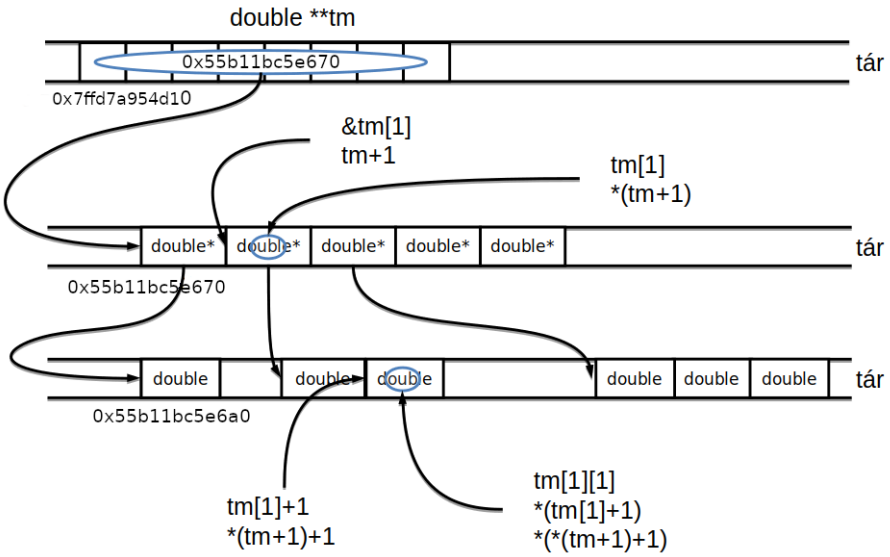
Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/haromszog.c

A mátrix n darab oszlopból és m darab sorból álló táblázatok. Egy mátrix akkor háromszögmátrix, ha a főátlója alatt vagy felett csak 0 található. A "malloc" függvénnyel memóriát foglalunk le, mi adjuk meg mekkora ez a terület. Ez után vagy a lefoglalt memória kezdőcímét vagy NULL értéket ad vissza. Egy for ciklussal végigmegyunk a lefoglalt teruleten és minden double tipusú teruletre állítunk egy mutatót, majd ezek a mutatók alkotják a háromszögmátrixot. A "free" függvénnyel felszabadítjuk a lefoglalt memóriát.



Deklarálunk egy ’double **tm’ változót. A ’double’ változók 8 bájtosak. A ’double **tm’ változó egy ’double *’-ra mutat, vagyis annak memóriacímére. A ’tm+1’ azt jelenti, hogy a ’tm’-nél eggyel arréb mutat, vagyis 4 bájtal arréb. A ’*(tm+1)’-el a benne lévő értéket kapjuk.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megololdás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/exor_titkosito_c.c

A kizáró vagyok titkosítást fogjuk használni. Ez a művelet amikor egymáshoz hasonlítjuk a szöveg bájtját és a titkósító kulcs bájtjait akkor 1-et ad vissza ha a kettő különböző és 0-t ha megegyeznek. A program elején megadjuk a titkosítani szánt szöveg méretét, ez alapján a program generál egy megfelelő méretű kulcsot. Ezután a kettő összehasonlításával megalkotja a titkosított szöveget.

Futtatás:

```
gcc exor_titkosito_.c -o exor
./exor 42351234 *<*tiszta.txt '>'titkos.txt
```

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:https://github.com/Drcsonka/Prog1/blob/master/ExorTitkosito.class https://github.com/Drcsonk

Ez a feladat az előző Exor titkosítás csak Java programozási nyelven.

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/Drcsonka/Prog1/blob/master/exor_toro_c.c

Ez a program az előbb létrehozott titkosítóra épül. A titkosított fájl-t összeveti a kulcsal és így visszakapjuk az eredeti szöveget.

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: https://github.com/Drcsonka/Prog1/blob/master/main.cpp

A Mandelbrot feladatból kapott képet használjuk fel bemenetként. Ezt a 'png::'-vel kezdődő sorban állítjuk be, és majd az első parancssori argumentumban adjuk meg. A 'get_width' és a 'get_height' szorzatából kapjuk meg a képünk méretét amit el is fogunk tárolni. A 'Perceptron *p' változóban létrehozunk egy új perceptront aminek 4 paramétert kell megadnunk. Az első a rétegek száma (3) , a második a 'size' vagyis az első rétegre annyi neutront akarunk (size) , a harmadik 256, vagyis a második réteg neuronjai (256), a negyedik az eredmény (1).

Két egymásba ágyazott 'for' ciklussal végigmegyünk a kép szélességén és magasságán. Eközben az 'image' változóban tárolódik a kép vörös színkomponensei. A Perceptronunkat meghívjuk az 'image'-re és megadjuk a 'value'értékét, ezzel egyben a perceptronba is tárolásra kerül a vörös színkomponens. A 'value' egy double típusu változó, amit kiírunk. A 'delete()' fügvénnyel felszabadítjuk a Perceptronunk által foglalt memóriát.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/mandel.cpp

Benoit Mandelbrot matematikus találta meg ezt a komplex számsíkot és róla lett elnevezve. A program futtatásával fogjuk elkészíteni a Mandelbrot-halmaz kétdimenziós, számítógépes ábráját. Meghatározzuk a szükséges adatokat a számításhoz: magasság, szélesség, iterációs határérték. "For" ciklusokkal megyünk végig a szelessegen, magassagon és kiszámítjuk az adott csomóponthoz tartozó komplex számot. Eközben egy while ciklussal az iteráció értékét számoljuk, és ha ez eléri a határt akkor a szám eleme a Mandelbrot-halmaznak.

Futtatasa:g++ mandelpngt.c++ -lpng16 -o mandelpngtki ./mandelpngtki t.png

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

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/3.1.2.cpp

Az elozo feladatban a komplex számokat két változóban tároltuk egyben a valós, másikban a képzetes részét, viszont az std::complex osztállyal megtehetjuk azt, hogy egy változóban tárolhassuk.

A while ciklussal azt nézzük meg, hogy mennyire távolodik el z_n a z_0 ponttól vagyis a koordinátarendszer közepétől. Ha ez a távolság nagyobb mint 2, akkor a vizsgált pontban az iteráció nem sűrüsodik be az origohoz kozeli pontban. Ha az elérjük az iterációs határt es emiatt lépünk ki a while ciklusból, akkor ezt a pontot feketére szinezzük, és a Mandelbrot-halmaz elemének tekintjük.

5.3. Biomorfok

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

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

Az Mandelbrot-halmaz-hoz szorososan kapcsolódik a Biomorf. Itt viszont A komplex iteráció a Mandelbrot-halmaz-ban egy változó volt, itt viszont állandó. Minden rácsponthoz más értéket vesz vesz az előző feladatokban, a mostaniban viszont ugyan az minden rácspontra.

A Mandelbrot-halmazhoz képest nemcsak a formáján különbözik hanem, ráadásul színes képet is generálunk. Ez azért lehetséges mivel a Mandelbrot-halmazhoz képest más képletet is használunk. Ez úgy valósul meg, hogy az iterávió számát elosztjuk maradékosan 255el, így ebből kapunk egy RGB kódot amivel színezzük az adott pontot.

Forditas: $ g++ biomorf.cpp -lpng -O3 -o biomorfki $ ./biomorfki biomorf.png 800 800 10 -2 2 -2 2 .285 0 10

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: https://github.com/Drcsonka/Prog1/tree/master/nagyito

A feladat megoldásához QT-t fogunk használni, mivel egyszerűen tudunk grafikus felületen programokat létrehozni.

A feladatunk 5 fájlból / részből fog összeállni.

Az első rész a 'main.cpp' lesz.

A Qt-t fogjuk előkészíteni itt, meghívni a hozzá szükséges könyvtárakat. A 'Q Application'-al példányosítjuk, és majd meghívásra kerül a konstruktor.

A második rész lesz a 'farkablak.h'.

Ez egy header fájl, amit a 'main.cpp'-hez kapcsolunk. Ebben a header fájlban fogjuk deklarálni a billentűlenyomás, egérmozgás és kattintáshoz használatához szükséges függvényeket.

A harmadik rész a 'frakablak.cpp' fájlban található.

Itt a 'farkablak.h' fájl által deklarált függvényeket fogjuk definiálni.

A negyedik rész a 'farkszal.h'.

Ebben a header fájlban fogjuk deklarálni a számoláshoz és rajzoláshoz használt változókat.

Az ötödik és egyben utolsó rész a 'farkszal.cpp' fájlban lesz.

Itt fogjuk a Mandelbrot-halmaz-t megrajzolni. Végighaladunk a szélességen és magasságon. Minden pontot elemezve, hogy része-e a Mandelbrot-halmaznak.

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: https://github.com/Drcsonka/Prog1/blob/master/polargen.cpp https://github.com/Drcsonka/Pro

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!

A feladatot 'C++'-ban és Javaban is megoldjuk. Kezdve 'C++'-al.

Először is csinálunk egy osztályt, itt fog elhelyezkedni a Konstruktor és a Dekonstruktor is aminek a neveinek meg kell egyeznie az osztályunk nevével. A konstruktor objektumok létrehozásához kell majd míg a Dekonstruktor az ezek törléséhez.

Az osztályban található még a 'tarolt', 'nincsTarolt' változók és a 'kovetkezo()' függvény is. A 'kovetkezo()' függvény egy double típusú értéket fog visszaadni. A 'nincsTárolt' egy boolean típusú változó ami azt fogja nekünk jelezni, hogy van-e eltárolva számunkra kiszámolt szám.

Amennyiben a 'nincstárolt' hamis, a program az eltárolt elemet adja vissza a függvénynek. Ha viszont igaz akkor, egy do-while ciklusban, megnézzük, hogy 'w' nagyobb-e mint 'u1' és 'u2', ha igen kap egy random számot a 'srand()' függvénytől, eközben a 'v1' és 'v2' megkapja 'u1' és 'u2' értékét. A 'w' új értéket kap a 'tárolt'-al együtt. Majd a 'nincstárolt' értéke hamis-ra változik mivel már nem fog elemet tartalmazni.

A feladat Java-ban nagyon hasonló és könyebb is mint a C++-os megoldás.

Itt is létrehozunk egy osztályt aminek a neve 'PolárGenerátor' lesz. Ez egy publikus osztály ami azt jelenti, hogy a benne lévő egységeket az osztályon kívülről is meg lehet hívni. Itt deklaráljuk a 'nincsTárolt' változót, és a 'következő()' függvényt is, ugyan úgy mint C++-ban. A 'nincsTárolt'-os ciklus is ugyan az mint C++-ban ugyan az lesz, a random szám generáláson kívül. Javában a 'Math.random()' függvény segítségével nagyon könnyedén tudunk random számokat generálni.

6.2. LZW

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

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/lzw.c

Ezzel a feladattal már foglalkoztunk Bev. Prog. órán. A program lényege, hogy a bemeneti fájl tartalmát átalakítja és kiírja a bináris fáját egy külön fájlba. A fa építés elve, hogy amikor a programunk kap egy 1-est vagy 0-ást megnézi, hogy az aktuális csomópont tartalmazza-e azt az elemet. Ha nem betesszük a jelenlegi csomópont gyermekeként. Amennyiben viszont igen akkor csinálunk egy új csomópontot és annak lesz a gyermeke.

A programban az 'uj_elem()' függvényben fogjuk lefoglalni a memóriaterületeket, de hibával tér vissza ha nem sikerül elegendő memóriát foglalni. A 'malloc' függvény segítségével fogunk memóriát lefoglalni, ennek paramétere a 'BINFA' változó lesz.

A 'szabadit()' függvénnyel fogjuk felszabadítani a jelenlegi fa bal- és jobb oldala által foglalt memóriát. A 'free()' függvénnyel a lefoglalt tárterületet szabadítjuk fel.

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/lzwpost.c https://github.com/Drcsonka/Prog1/

A fabejárás előtt megnézzük, hogy a bejárandó fa üres-e. Preorder bejárásnál elhelyezzük a gyökérelemet a sor végére. Majd először a gyökérelem bal oldali részfáját, ezután a jobb oldali részfáját járjuk be.

Postorder a Preorder ellentéte, ilyenkor először a gyökérelem bal oldali részfáját, majd a jobb oldali részfáját járjuk be, és csak ezután dolgozzuk fel a gyökérelemet.

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/z3a7.cpp

Ebben a feladatban megint az eddig használt LZW binfa programot fogjuk használni. Csak C++-ban írva és nem C-ben.

Az első változás abban lesz, hogy itt már használhatunk osztályokat. Az osztályokban nemcsak változókat, hanem függvényeket is írhatunk. Az osztályokban deklarált függvényeket az osztályon kívül definiáljuk.

Még egy érdekesség az operátor túlterhelés ami segítségével az módosítani tudjuk, hogy egy operátor mit csináljon. A programban arra használjuk, hogy a bemenetként kapott elemeket a fába shifteljük, az 'LZW' algoritmusnak megfelelően.

Ha a programunk 0-at kap, akkor megnézzük, hogy az jelenlegi csomópontunknak van e 0-ás gyermeke. Ha nincs, akkor először a 'new'-val lefoglaljuk a tárterületet és egy új csomópontot hozunk létre. Ezután az aktuális csomópont nullás gyermekét ráallítjuk az új csomópontra és a fával visszaállunk a gyökérre.Ha viszont van nullás gyerkmek, akkor a fa mutatóját állítjuk rá. Ha 1-est kapunk akkor is ugyan ez történik.

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: https://github.com/Drcsonka/Prog1/blob/master/z3a72.cpp

Az előző feladathoz képest az fog most különbözni, hogy most a gyökér is mutató lesz. Most mindenhol, ahol eddig a gyökeret referenciaként adtuk át a mutatónak, most enélkül fogjuk. Helyet foglalunk a memóriában és erre állítjuk rá a fát.

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/blob/master/z3a73.cpp

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://github.com/Drcsonka/Prog1/tree/master/hangyaszimulacioé

Tanulságok, tapasztalatok, magyarázat...

Az Ant.h header fájlban a hangyák tulajdonságai vannak deklarálva. Az 'x' és 'y' változók a koordinátái, hogy hol vannak éppen, a 'dir' változó pedig azt mutatja, hogy milyen irányba tart éppen.

Az AntWin.h header-ben az ablak magassága és szélessége van megadva pixelekben ez publikus részben van megadva. A 'cellwidth' és 'cellheight' változók pedig a cellák szélességét és magasságát adják meg amiben a hangyák vannak megjelenitve ez privát részben vannak deklarálva. A 'grids' a két rácsot adja meg, a 'gridldx' pedig a két rácspont közül az egyiket tárolja.

A 'closeEvent()' függvény meghívja az AntThread.h header-ből a 'finish()' methodust ami leállítja a programot ha hamis értéket kap.

A 'keyPressEvent()' függvény a gomb lenyomásokat dolgozza fel.

A 'paintEvent()' függvénnyel pedig megváltoztatjuk a hangyák színét.

Az 'AntThread' osztály nagyon hasonló az 'AntWinhez' csak kiegészítve újabb tulajdontságokkal. Ezek például az 'evaporation' ami a párolgás mértékét tárolja el. Az 'nbrPheromone' ami a feromonok számát. A 'cellAntMax' a hangyák maximális előfordulását egy cellán belül.

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/Drcsonka/Prog1/blob/master/Sejtautomata.java

Tanulságok, tapasztalatok, magyarázat...

A program alap helyzetben üres rácsokkal van tele. Kiinduláskor létrehozunk egy általunk kitalált alakzatot, és ebből fog elindulni a játék. Az alakzat rácsai élő sejtek lesznek, a többi cella halott sejtek. Egy sejtet 8 cella vesz körül. Ahhoz, hogy egy élő sejt tovább éljen legalább 2 vagy 3 szomszédos élő sejtjének kell lennie. Egy halott sejt akkor kel életre, ha 3 vagy több közvetlen mellette lévő sejt elő. Két rácsot használunk, az egyik a sejttér jelenlegi állapota, a másik a sejttér megváltozott állapota. Ezek az 'időFejlődés()' függvényben vannak. Itt figyeljük, hogy a sejtek hogyan változnak.

A 'szomszedokSzama()' függvényben megnézzük, hogy a sejteknek mennyi szomszéda van, és ez alapján, hogy mi fog történni vele. Definiáljuk, hogy hány cella magasa és széles legyen a sejtterünk és, hogy a cellának mekkora legyen a mérete. Hogy mennyi legyen a setterek közötti váltakozás ideje.

A paint() függvény kirajzolja számunkra a sejtteret és az azon lévő sejteket. Az egybe ágyazott for ciklusból a külső felel a sejttér soraiért a belső az oszlopaiért. Az élő sejteket feketével a halottakat viszont fehérrel rajzolja ki a program számunkra.

Gombokkal tudjuk a játék menetet valamilyen szinten változtatni. Az 'S' betű lenyomásával pillanatképet tudunk csinálni amit számolunk is, hogy hány készült. A szimuláció sebességét is tudjuk változtatni a 'g' és az 'l' billentyűkkel. A 'g'-vel gyorístjuk az 'l'-el lassítjuk a rácsok váltakozása közötti időközt. A cella méreteit a 'K' és az 'N' billenytűvel tudjuk változtatni. A 'K' betűvel felezzük az 'N'-el duplázzuk a sejtek méreteit. Az egerérre is reagál a program. Ha rávisszük az egerünket egy cellára akkor az elővé változik, ha viszont lenyomjuk a klikket akkor a cella jelenlegi állapotának az ellenkezőjére változik.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/tree/master/sejtautomata_cpp

Ez a feladat ugyan az a program lesz mint az előző feladatban csinált életjáték, csak ebben az esetben c++-ban QT keretrendszer segítségével.

A 'sejtablak.h' header fájlban deklaráljuk a program alapját. A sejttér magasságát ('magassag') és szélességét ('szelesseg') vagyis a nagyságát. A 'cellaSzelesseg' és a 'cellaMagassag' a cella adatait adják meg pixelben. A két rácsot is itt deklaráljuk ami a sejttér két időállapotot fogják tartalmazni. Ezen felül még azt is, hogy a sejt él-e vagy sem.

A 'sejtszal.cpp' file-ban több függvényt is deklarálunk. Az egyik a 'szomszedokSzama()' függvény amiben végignézzük a sejtnek a szomszédait. A másik az 'idoFejlodes()' függvény amiben azt határozzuk meg, hogy a sejtünk túl fogja-e élni ezt az időállapotot, vagy sem. Akkor éli túl az élő sejt ha 2 vagy 3 élő szomszédja van, amúgy meghal. A halott sejt viszont marad halott, hacsak nincs 3 élő szomszédja, mert akkor az is élő lesz.

A 'sejtablak.cpp' fájlban a 'paintEvent()' függvényt deklaráljuk ami az aktuális rácsot rajzolja ki. 2 'for' ciklus rajzolja ki a sejtcellát, a belső 'for' ciklus az oszlopokat, a külső a sorokat. Ezaltal minden sejtet kirajzolnak a megfelelő pontokra.

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: https://github.com/Drcsonka/Prog1/tree/master/brainH

Ezt a programot arra a célra fejlesztettékm hogy az esportolókat teszteljék mennyire tudják követni a karakterük mozgását zavarosabb helyzetekben.

A program alapja, hogy a karakterünket vagyis egy pontot kell követnünk az egerünkel, és ha egy másodpercnél tovább nem tartozkodik az egerünk a dobozban akkor, a program annak veszi, hogy elvesztettük a dobozt és lassul a doboz mozgása. A játék során egyre több doboz fog megjelenni a képernyőn ezzel nehezítve a 'karakterünk' követését.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Megoldás videó:

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

Tanulságok, tapasztalatok, magyarázat... A TensorFlow egy google álltal készített és használt, mesterséges intelligenciara is felhasznalhato szoftver. A most hasznalt programunk egy 28 x 28 pixeles képet fog kapni és az azon szereplő számot kell felismernie ('readimg' függvény fogja beolvasni ezt a képet). Ahhoz, hogy a keprol fel tudja ismerni a szamot, meg kell ra tanitanunk a programot. Ezt többszöri futtatassal csinaljuk, minden egyes lefutasnal kiiratjuk a pontossag értékét, hogy mi volt a képen szereplő szám, és, hogy a program mit tippel.

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... (passzolva)

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... (passzolva)

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

A Lisp abban különbözik a többi programozási nyelvtől, hogy prefix alakban kell megadni a műveleteket vagyis nem a megszokott (1 + 1) alakban, hanem (+ 1 1) alakban. Függvényeket viszont a define parancsal definiálunk. Az iteratív faktoriális lényege, hogy amennyi a megadott szám annyiszor végzünk el szorzást, és ezzel érjük el a faktoriálisát.

```
(define (fact n) (do (( i 1 (+ 1 i)) (num 1 (* i num))) ((> i n) num)))
```

A rekurzív faktoriálisnál viszont n-1 faktoriálist megszorozzuk az n-el, és ezt addig csináljuk ameddig a számunk vagyis n egyenlő lesz 1-el vagy kisebb.

```
(define (fact n) (if(< n 1) 1 (* n (fakt(- n 1)))))
```

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... A Lisp programozási nyelvek családja közé tartozik a Scheme is. Maga a programozási nyelv régi de egyszerűsége miatt még most is használják. A feladatunk az lenne, hogy a 'Gimp' nevű ingyenes képszerkesztő programban a Script-Fu-val elerjük, hogy a kívánt szövegünknek 'Króm' effektje legyen. A 'script-fu-bhax-chrome-border' függvény fogja létrehozni a króm effektes szöveget. Ezt úgy éri el, hogy egy új rétegre teszi, aminek fekete lesz a háttere és a rajta lévő szöveg fehér.

A script fájl elhelyezése után, a 'Létrehozás' menüpontnál lesz egy új opciónk a: Chrome3. Ezt kiválasztva kapunk egy menüt. Amit itt láthatunk:

```
(script-fu-register "script-fu-bhax-chrome"
    "Chrome3"
    "Creates a chrome effect on a given text."
    "Norbert Bátfai"
    "Copyright 2019, Norbert Bátfai"
    "January 19, 2019"
    ""
    SF-STRING       "Text"      "Bátf41 Haxor"
    SF-FONT         "Font"      "Sans"
    SF-ADJUSTMENT   "Font size" '(100 1 1000 1 10 0 1)
    SF-VALUE        "Width"     "1000"
    SF-VALUE        "Height"    "1000"
    SF-COLOR        "Color"     '(255 0 0)
    SF-GRADIENT     "Gradient"  "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
    "<Image>/File/Create/BHAX"
)
```

Ebbena menüben 'Text'-hez a kívánt szöveget írjuk, amit króm színben szeretnénk látni. A többi menűpont a betűtípust ('Font'), betűtípus méretét('Font size'), szélesség('Width'), magasság('Height'), szín('Color') amit RGB színkódban adunk meg és a csillogás típusát('Gradient').

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... Ez a feladat is Gimpben Scheme programozási nyelven Scirpt-fu-ban fog írüdni. A feladat az lenne, hogy egy mandala készüljön az általunk megadott szövegből. A mandala egy kör alakú alakzat amit szöveg forgatásával kapunk. Itt is ugyan úgy fog működni a menüje, mint az előző feladatban a króm színnél.

És itt is ugyanúgy fogjuk meghatározni a szöveg méretét:

```
(set! text-width (car (gimp-text-get-extents-fontname text fontsize PIXELS ↩
   font)))
    (set! text-height (elem 2  (gimp-text-get-extents-fontname text ↩
       fontsize PIXELS font)))
```

Majd létrehozunk egy új réteget, amire ráillesztjük a megadott szöveget. A mandala képet úgy éri el a program, hogy fogja a szöveget, tükrözi, majd elforgatja es ezt fogja ismételni.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

1.2

Megismerjük a programozási nyelvek 3 szintjét. Ezek a gépi nyelv, az assembly nyelv és a magas szintű nyelv. A könyvben a magas szintű nyelvvel fogunk jobban foglalkozni, ezeket forrásprogramnak is szokták hívni. A processzorok nem tudják értelmezni a forrásprogramokat ezért szükség van fordítóprogramokra, amik gépi kódú programot készít. Ennek lépései :Lexikális elemzés, szintaktikai elemzés, kódgenerálás. Megismerjük ezen kívül a szintaktika és a semantika fogalmait.

1.3

A nyelveket 2 nagy részre lehet bontani, ez a kettő az Imperattív nyelv és a Dekleratív nyelv.

Az imperatív nyelv jellemzői: Algoritimikus nyelvek, utasítások sorozatára épülnek fel, szorosan kötődnek a Neumann-arcthitektúrához, legfőbb eszköze a változók. Két alcsoportja van az objektumorientált nyelvek és az eljárásorientált nyelvek.

A dekleratív nyelvre jellemző: Nem algoritmikus nyelv, nincs lehetőség memóriaműveletekre. Két alcsoportja: a logikai nyelv és a funkcionális nyelv.

2.1

A program legkisebb alkotórésze a karakter. Általában a karaktereket kategorizálják, betűknek, számjegyeknek vagy egyéb karaktereknek. Ide tartoznak a szimbolikus nyelvek is. Ennek két része van a kulcsszó és az azonotító. Az azonosító a felhasználó számára van, hogy arra ezzel tudjon hivatkozni az előző elemekre. A kulcsszó viszont az nyelvnek van, és ezt felhasználóként nem tudjuk megváltoztatni.

2.4

A programnyelveknek vannak adatípusaik is. Ezek lehetnek a programozási nyelv által megadott típusok vagy a felhasználó is hozhat létre típusokat, ilyenkor meg kell adni a típus tartományát, műveleteit és ábrázolási módját. Az adattípusoknak két nagy csoportja van, az egyszerű és az öszetett típus. Ezekből egy pár példa: az egész számok, tömbök, listák, karakterek, logikai.

2.4.3

A mutató típus egy egyszerű típusu adattípus. Tárcímeket tárol. Ha nem mutatnak sehova akkor 'NULL' értékre mutatnak. Velük tudunk indirektül címezni.

2.5

Nevesített konstansoknak 3 komponesből állnak, a névből, típusból és értékből. Ezt a konstanst mindig deklarálni kell. A deklarációnál kap egy értéket ami megváltoztathatatlan lesz később. Az a feladatuk, hogy a rendszeresen használt értékeknek egy más / könyebb nevet ad.

2.6

A változók 4 komponensből állnak, a névből, attríbútumokból, címekből és értékekből. A név az egy egyedi azonosító, az attríbútum a változó futás közbeni működéséről felel. A változóknak érétéket rendelhetünk aminek két fajtája van, az explicit, implicit és automatikus deklaráció. A változó címe a változó helyét határozza meg. Ezt megtehetjük több módon is, például statikus kiosztás, dinamikus kiosztás, és a felhasználó által kiosztás. Az érték komponense a bitkombináiót jelenti ami a címen helyezkedik el.

2.7

C-ben az alapelem típusáról van szó. Ez a kettő az aritmetikai típus, a számraztatott típús és a void típus.

3.

A kifejezések arra jók, hogy a már ismert értékekből új értéket csinálunk. Ennek két része van: az érték és a típus. Ezen kívül a zárójelezés menetét éritni, a konstans kifejezéseket és a tömbökről.

4.

Az utasításoknak két nagy csoportja van a deklarációs utasítások és a végrehajtható utasítások. A végrehajó utasításokból nagyon sok van, például a cikluszszervező, hívó, értékadó, üres, ugró stb.A ciklusszervező utasítások. Egy ciklus 3 részből áll, a fej, mag és a vég. A magban található az ismételendő kódrész. Az értékadó utasítás feladata például egy változó értékkomponensének beállítása.

5.

Egy program programegységekre tagolható amik az alprogram, blokk, csomag és a task. Az alprogramnak a használatára és fogalmáról beszél még. Az alprogram felépítése : Fej, specifikáció, törzs és a vég. A komponensei: név, formális paraméterlista, törzs, környezet. Az alprogramokat elég csak egyszer megírni és később csak elég hivatkozni rájuk. Két fajtája van: Eljárás és Függvény. Az függvény tetszőleges típusu eredményt ad vissza, viszont az Eljárásnak nincs visszatérési értéke de tevékenységet hajt végre.

5.4 Paraméterkiértékelés függvény hívásánál kiértékeli az alprogram formális és aktuális paramétereit. Az aktuálist rendeljük hozzá a formálishoz, és olyan elven, hogy az elsőt az elsőhöz... stb.

5.5

A paraméterátadásnak több módja is van: érték szerinti, név szerinti, cím szerinti, szöveg szerinti, eredmény szerinti, érték-eredmény, szerinti.

Később a blokkról van szó. A fogalmaráról formalításáról. A hatáskör fogalma, név hatásköre és a lokális név fogalma.

A nyelvek Input, Outputja nagyon különböznek. Említik a fogalmát, és hogy állományokba lehet írni, és ezekből tudunk dolgozni is.

Kívételkezelés az az amikor a program figyel egy bizonyos eseményt és annak bekövetkezésére, reagál / csinál valamit.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

Az alapokról van szó a könyv elején. Például változótípusok: char (1 byte, egy karaktert tartalmaz), int (egész szám), float és a double. Különböző operátorok mint pl: +,-,*,/,%. A logikai operátorok és relációk.

Vegyünk például az 'if' blokkot. Ez például áll a fejből 'if', ez után áll egy logikai rész, ami ha igaz akkor a benne lévő kód leful, amugy meg csak átlépődnek azon a kódrészleten és sosem fut le. If-else esetében viszont az 'if' résznél megadunk egy logikai kérdést, ha igen akkor lefut a kódrészletünk az 'if' alatt, ha viszont hamis akkor az 'else' rész utáni kód fog lefutni.

A ciklusok például: for, while, do-while. A while ciklus úgy müködik, hogy addig fog végrehajtódni ameddig a fejében lévő kiértékelés 0 nem lesz. A for ciklus 3 kifejezést tartalmaz a fejben. Az első hogy mettől, a második, hogy meddig, a harmadik, hogy milyen közzel. Es ez addig meg még az első érték el nem éri a második értéket.

Több féle utasítás is van ezek például a: while, for, break, continue, return.

A break utasítás ciklusokba lehet implementálni, és ez leállítja a ciklusunkat, amikor arra kerül a sor.

A return utasítás visszaad egy értéket.

C-ben lehet címkére ugrani a goto utasítással.

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

10.3. Programozás

[BMECPP]

A C++ egy objektum orientált programozási nyelv mint a java. A könyv megmutatja a C és a C++ közötti változtatásokat és különbségeket. C++ például két függvénynek lehet ugyan az a neve ha az argumentumai különböznek. C++-ban függvényt meglehet hívni paraméter nélkül ilyenkor voiddal lesz egyenlő. Bevezetésre kerültek a refereciák és a bool változó is ami egy logikai érték, így igaz / hamis értéke lehet. Pointerekről is szó esik.

Megismerjük az objektumorientáltság alapjait. Az objektumokat bevezeti az öröklés fogalmával együtt. Az objektum az egy osztály egy előfordulása.

A 'private' kulcsszó azt csinálja, hogy az alatta deklarált változók és függvények csak benne láthatóak és kívülről nem meghívhatóak. Ezzel ellentétben a 'public' alatt deklarált függvények és váltózók az adott tagon kívül is látszódnak.

A konstruktor és a dekonstruktor előre definiált függvények. A konstruktor biztosítja, hogy az, a feladathoz legyen elég nagy méretű tárterület számára, és legyen kezdeti értéke. A dekonstruktor viszont ezt a lefoglalt területet szabadítja majd fel, vagy legalábbis segít benne.

Operátor túlterhelésről is szó esik. Az operátorokról úgy általánosságban. Az operátorokra tekinthetünk speciális függvényekként is.

Statikus tagok nem objektumokhoz hanem osztályokhoz tartoznak. Ezek objektum nélkül is használhatóak.

A szabványos adatfolyam kicsit más C++ alatt, mivel itt bejön az 'istream' és 'ostream', vagyis az adatfolyamok bemenete és kimenete. Ezeknek operátoraik a >> és a forditott kacsacsőr duplán.

A kivételkezelés abban segit nekünk vagy legalábbis azt biztosítja, hogy ha hiba merül fel akkor a program azt kezdi el kezelni azonnal.

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.