Tanuljunk együtt programozni!

Írd meg a saját programozás tankönyvedet!

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

COLLABORATORS

	TITLE : Tanuljunk együtt programozni!	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Jánócsik, Csaba	May 10, 2019	

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]

Contents

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.

Part I

Bevezetés

Chapter 1

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.

Part II

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.

Chapter 2

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: 1magszázon: https://gist.github.com/AmidaHS/731835bde99e9ca5665f1e3f6fd5337b#file-egymagszaz-c összesmagszázon: https://gist.github.com/AmidaHS/cef6c1eae814cbdc26593074ea0aa7c0#file-osszesmag-c 1 mag 0%-on: https://gist.github.com/AmidaHS/f9c09cfe39cebdf80eb0f9f59585e04c

Tanulságok, tapasztalatok, magyarázat... 1mag 100%: Írunk egy alapvető végtelen ciklust,a feltétel mindig teljesül.Így a használt mag mindvégig dolgozni fog. összes mag 100%: Előzőhöz hasonló annyi különbséggel, hogy a #pragma omp parallel -el párhuzamosítjuk és így egyszerre több magon fogja ugyanazt elvégezni. 1mag 0%: Az előzőekhez hasonló, kivétel hogy a sleep függvény segítségével altatjuk a programot, ezálltal futni fog, de nem használja a cpu-t.

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... Egy paradoxont kapunk, mivel nem létezik olyan program ami eltudja dönteni egy programról hogy lefagy-e, mivel ha a program tartalmaz végtelen ciklust, akkor végtelen ideig tudná vizsgálni is.

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://gist.github.com/AmidaHS/9fef6c7e5ddeb6df52e6e59803c3a03a

Tanulságok, tapasztalatok, magyarázat... Ennek a feladatnak a megoldásához csupán összeadni és kivonni kell, semmi nehezebb programozás nincs benne.Az elején bekérünk 2 egész számot. Majd a számításokat elvégeztetjük.

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:

if-fel: https://gist.github.com/AmidaHS/b10c0ad17f7d8519a61cb8d8fb8d68e7

if nélkül: https://gist.github.com/AmidaHS/74d6abc5206e0365597ff246ed252522

Tanulságok, tapasztalatok, magyarázat.. IF-fel:A program során megadjuk a képernyőtávot ahol a program lefuthat, a labda helyzetét.Írunk egy végtelen ciklust, ami adja meg a "labdapattogást". A usleep segítségével fogjuk a labda sebességét megváltoztatni,míg a refreshel a képernyőt fogjuk "tisztán" tartani. Az if-es sorok fogják megadni hogy a labda hogyan pattanjon egyik falról a másikra. IF nélkül: Hasonló annyi különbséggel, hogy itt nem az if-fel hanem a mvprintw segítségével deklaráljuk le a mozgást is.

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://gist.github.com/AmidaHS/d148a9fe32cdcd71c960583fccb841da

Tanulságok, tapasztalatok, magyarázat... A számláláshoz bitshiftelést használtam. Bővebben:(Forrás:Wikipédia: https://hu.wikipedia.org/wiki/Bitm%C5%B1velet) A biteltolást (bit shift) is szokás bitműveletnek tekinteni, mivel nem számértékeken, hanem bitek sorozatán dolgozik. A biteltolás során a bináris számjegyek balra vagy jobbra tolódnak el („shift”-elődnek). Mivel a processzor regiszterei fix szélességűek, ezért egy vagy több bit ki fog shiftelődni a regiszter valamelyik végén, a másikon pedig ugyanannyi bit beshiftelődik; a biteltolási műveletek közötti különbséget épp az adja, hogy a beshiftelődő bitek értékét honnan vesszük.

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://gist.github.com/AmidaHS/d186c35ffaac9db8e25a97fe64458633

Tanulságok, tapasztalatok, magyarázat... Ezt a Page Rank rendszert a Google alapítói találták ki Larry Page és Sergey Brin. Ennek a rendszernek a lényege, hogy egyfajta "szavazásra" épít, egy weboldalra minél több hiperlink tartozik annál jobbnak minősül, de ez oda-vissza működik, azaz aki "leadja a szavazatot" arra hányan szavaztak. Ez alapján állít fel egy fontossági sorrendet. Többek között ennek köszönheti a Google a sikereinek titkát.

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

Tanulságok, tapasztalatok, magyarázat... A brun tétel Viggo Brun nevéhez fűződik, aki 1919-ben bizonyította ezt be.Tételének lényege az ikerprímekhez köthető, azaz olyan prímpárokat keresett amelyeknek különbsége 2.Rájött hogy ezen prímek reciprokösszege egy később "Brun-konstans"-ként elnevezett ismert véges értékhez konvergál. Az R nyelves programunkhoz szükség van először is a matlab függvénykönyvtárra,ezuta létrehozzuk az stp függvényünket. Programunk bekér egy paramétert, majd a matlabos primes függvénynek köszönhetően az adott értékig kiírja az összes prím számot.A diff résznél azt fogjuk tárolni, hogy a prímek között mekkora érték különbség található.Az idx-ben pedig az előző értékekből keresi azokat amelyeknek a különbsége 2.Ezeket megkeresve már csak a Brun-tételt kell alkalmazni. Utána már csak azokat az adatokat adjuk meg amelyek ahhoz kellenek, ami a matlab általl nyújtottak, ahoz hogy kirajzolódhasson a függvényünk.Pl:Megadjuk x és y értékét, milyen lépésközzel haladjunk, vagy maga a grafikon kiiratása a plot segítségével hogyan történjen.

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... A probléma, hogy létezik három választási lehetőség, egy nyertes valamilyen nagy értékkel még kettő üres, vagy valami értéktelen tárgyal. A játékos nyereménye a választól függ, de először a játékos csak választ és ahelyett, hogy véget érne a játék a műsorvezető a két megmaradt lehetőség közül megmutatja az egyik vesztest. A játékvezető tudja melyik nyer és veszít. Ezután megkérdezi a játékost, hogy akar-e változtatni. A játékos ezután vagy változtat, vagy nem. Ezután a játékos megtudja, hogy mit rejt a választása. A paradoxon az, hogy érdemes-e változtatni, illetve hogy számít-e ez egyáltalán? És igen érdemes mivel valószínűség számításból kiindulva ha 3 lehetőség közül választunk egyet akkor abban a pillanatban mindegyik válasz 1/3 esélyt rejt arra, hogy nyertes legyen. Így az összes lehetőség úgymond 1/3nak felel meg, de ezt követően mivel a másik két lehetőség közös változóként lép fel és az egyik kinyílik így annak a valószínűsége 0 lesz és az ő 1/3 átmegy a másikra amlyeiket nem választottuk ezáltál 2/3os nyerési esélyt kap. Az R nyelves programunk ezt az esetet bizonyítja a megadott számú ismétléssel a véletlenszerűséget felhasználva és a játékos egyéni döntései alapján.

Chapter 3

Helló, Chomsky!

3.1 Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása:

Az unáris számrendszerben való ábrázolás n darab (n a decimális szám) egyforma jel, karakter egymás utáni leírásával történik.

A decimálisból unárisba átváltás úgy történik, hogy folyamatosan 1-eket vonunk ki a számból, és tároljuk a levont egyeseket.

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

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

Megoldás videó:

Megoldás:

1:

```
Szabályok:
S -> aBSc
S -> abc
Ba -> aB
Bb -> bb
Példa levezetés:
S -> aBSc -> aBaBScc -> aBaBabccc -> aaBBabccc -> aaBaBbccc -> aaaBBbccc -> ↩
    aaaBbbccc -> aaabbbccc
```

2:

```
Szabályok:
S -> abc
S -> aXbc
```

```
Xb -> bX
Xc -> Ybcc
bY -> Yb
aY -> aaX
aY -> aa
Példa levezetés:
S -> aXbc -> abXc -> abYbcc -> aYbbcc -> aaXbbcc -> aabXbcc -> aabbXcc ->  ↩
   aabbYbccc -> aabYbbccc -> aaYbbbccc -> aaabbbccc
```

A generatív nyelvtan elméletét Noam Chomsky alkotta meg, és ő dolgozta ki a Chomsky-hierarchiát. A formális grammatikákának három típusát ismerjük, a környezetfüggetlen nyelvtan, a szabályos nyelvtan és a generatív nyelvtan. A környezetfüggetlen nyelvtanban a szabályok megadása esetén a sazbály bal oldalán csak nem terminális változó állhat, illetve a jobb oldalán csak terminális változók állhatnak.

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://gist.github.com/AmidaHS/51e3e1decafd8d80127310d1f9dcfeee

Tanulságok, tapasztalatok, magyarázat... A kódunk a "gcc -std=c89 nev.c -o nev" segítségével nem tud lefordulni, hiszen a hibaüzenetből kiderül, hogy a C++-os kommentelést nem támogatja. Ellenkező esetben, míg például a "gcc -std=c99 nev.c -o nev" -vel már gond nélkül működik a programunk.

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://gist.github.com/AmidaHS/9fc13b2e82ba699927762c5621fafa34

Tanulságok, tapasztalatok, magyarázat... Ennek a feladatnak a megoldásához először egy lexer-t kell írni. A programunk elején a %{ %} segítsétségével a majd későbbi c programunkhoz tehetünk részeket. Deklaráljuk, hogy mit is keresünk és hogy azt hogyan tegye(azaz, hogyan néz ki), jelen esetben a valós számokat.Ezután az yylex-el visszahívjuk az előzőeket és kiiratjuk a darabszámot. A futtatás során először C-be kell átírni (lex segítségével), ezután pedig a szokásos módon(gcc-vel), csak egy kapcsolóval ki kell toldani: -lfl ami a lexer miatt elengedhetetlen.

3.5 l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/b924fb862b98bf7855e8de444e32b06d

Tanulságok, tapasztalatok, magyarázat... A szokásos módon include-olunk, majd egy random számot generálunk.A cipher tartalmazza a leet kódunk számait és betűit. Működése: A cipher megnézi az összes beolvasott karakterre hogy a kódolandó karakterekben megtalálható-e.Ha igen, kiválasztja a karakterhez a kódolást, ha nem akkor az eredetit írja ki.

3.6 A források olvasása

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

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

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

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

---

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

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

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

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

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

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

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

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

Megoldás forrása: https://gist.github.com/AmidaHS/3919a161a8a16998b8e1cb07d18fd515

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

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

: For ciklus, ahol az i nulláról ameddig kisebb mint 5 növeljük 1-el.(pre inkrementáló)

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

: Hasonló az előzőhöz csak post inkrementáló.

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

:Bugos: kétszeri hivatkozás történik és mivel nem ismerjük a műveleti sorrendet kiszámíthatatlan.

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

:Bugos: rossz operátort (nem logikai) használunk a `&&` rész után.

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

:Bugos:Hasonlóan a kettővel ezelőttihez, nem tudjuk eldönteni a sorrendjét a két f függvénynek, így meghatározni se tudjuk.

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

:Kiírja az a és az f függvény általl megváltoztatott a értékét.

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

:Bugos: Hasonlóan az egyik előzőhöz, itt is a sorrendel van a baj, hiszen az f függvény megváltoztatja az "a" értékét, de így nem tudhatjuk, hogy a kiiratott "a" az új vagy az eredeti értékét adja-e vissza.

3.7 Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

3.8 Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

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

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

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/fb5d246953a2fc2c518c763df324c80a

Tanulságok, tapasztalatok, magyarázat...

```
int a;
```

egy egész vezet be a programba.

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

egy egészre mutató mutatót vezet be.

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

egy egésznek a referenciáját vezeti be.

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

egy 5 egészből álló tömböt vezet be.

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

az egészek tömbjének referenciáját vezeti be.

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

egy egészre mutató mutatók tömbje.

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

egy olyan függvény ami az egészre mutató mutatót adja vissza.

```
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 mutatót visszaadó, egészet kapó függvény.

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

függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre.
```

Chapter 4

Helló, Caesar!

4.1 double ** háromszögmátrix

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/736d72c4e4d8e61fbad3c487813bbb4f

Tanulságok, tapasztalatok, magyarázat...Ahhoz, hogy egy háromszögmátrixot kapjunk egy négyzetes mátrixnak kell venni a főátlóját majd vagy az alsó vagy a felső részt kell 0-vá tenni, jelen esetben a felső részt. A double típusú mutatót használjuk és a mallockal függvénnyel a paraméternek lefoglaljuk a szükséges helyett. Megadjuk, hogy hány bájtos lesz úgy, hogy a sorok számát szorozva 8al,majd rákényszerítjük, hogy double legyen, ha nem sikerül akkor kilép. Ha ez sikeres akkor, a program végig fut az összes soron és kiírja a sorok értékét. Ezt követően a program megadja az oszlopok értékét. Ezek után megnézzük, hogy a különböző mutatók, hogyan változtatják meg a háromszég mátrix különböző értékeit

4.2 C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/63f908a3985f40ce1b4fa7b49356e75f

Tanulságok, tapasztalatok, magyarázat... A programunk alapja hogy titkosítani akarunk egy szövegállományt.Ezt a bitek megváltoztatásával tesszük. Először a szükséges könyvtárakat kérjük be majd megadjuk a maximális méretét a kulcsnak és a tárolónak egyaránt.A mainben megadjuk a függvény argumentumát integerben,ennek az értékét a számát valamint a rámutató mutatókat(char-os rész). Ezután deklaráljuk a kulcs méretet és a beolvasott bájtokat, majd a a while ciklust 3 fő részre oszthatjuk: megadja a jelenlegi inputot azaz hol járunk, ezeket a bájtokat berakjuk a bufferbe, majd az olvasott bájtokat kapjuk meg. Végigmenve a bájtokon végrehajtjuk a műveletet a kulcs következő bájtjával. Ezt mindig növeljük 1-el, de az osztás miatt sose lesz a megadott kulcsméreténél nagyobb. Ezek után ha kiakarjuk ezt iratni, akkor egy számunkra imeretlen és olvashatatlan kódot látunk.

4.3 Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/f72bdf8fa8248dee6cba8310c2219281

Tanulságok, tapasztalatok, magyarázat... A program lényege ugyanaz mint az előző feladatban, annyi eltéréssel, hogy ez javaban lett megírva. A c nyelvtől eltérően a javaban osztalyokat avagy classokat használunk. Kétféle classt definiálunk privát és publikus. A kód továbbá hasonlít a c nyelvi parancsokhoz csak a java sajátosságait kihasználva. A program végénél találhatunk egy try-catch blockot ami a try utánirészeket végrehajtja és a catch után ha van hiba kiírja.

4.4 C EXOR törő

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

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/e4bdac6cd59ab52426f3a99db607668a

Tanulságok, tapasztalatok, magyarázat... Programunk célja az előző titkosításnak a visszakódolása. Az elején a már szokásosnak mondható include-olás és definiáljuk a szükséges elemeket.Ez után kiszámoljuk az átlagos szóhosszt, a szóközöket elvesszük a karakterek számából.Későbbiekben a leggyakoribb szavakkal csökkentjük a törések számát, ezzel könnyítve a szöveg visszakódolását. Az exor részben bájtról-bájtra haladva elvégezzük a törést. Érdemes figyelni a % használatára,mivel ennek segítségével nem számít ha nem akkora a szó mint a keresett kulcs. A while ciklusban addig olvassa a bájtokat míg el nem fogy.A for ciklusok segítségével előállítjuk az összes lehetséges kulcsot majd teszteli azokat, hogy megfelelőek-e. Hasonló a működése és a felépítése, mint a közismertebb brute force algoritmushoz.

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

Feladatunk egy egyszerű logikai műveletre épülő neurális háló létrehozása.Legnagyobb részében négy kód bekezdés ismétlődik,viszont mégis más a logikai műveletek végett. Úgy tudjuk megrajzolni a neurális hálónk képét, ha előtte megadjuk, hogy milyen eredményt kell elérnie. Ezt a kaott hálót lekérjük,ami kiszámolja a logikai műveletek eredményét is. Végül a komplexitás emelése miatt növeljük a műveletek pontosságát.

4.6 Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/b04d2b2effe48ee0d7c567f22bad557c

Tanulságok, tapasztalatok, magyarázat... Nézzük meg először mi és hogyan néz ki egy perceptron: A perceptron a neuron az MI-ban használt legelterjedtebb formája, amelynek legfőbb feladata egy véges számú próbálkozásból, hogy megtanulja értelmezni és osztályozni a bemeneti mintát(0 v 1). Felépítése a következő: Három fő részből áll:Retinák,asszociatív cellák és döntési cellák.A retinák fogadják a bemeneti jeleket, amelyek általában 0 vagy 1(igen vagy nem) választ adnak.Az asszociatív cellák az összekötő kapcsok a retinák más asszociatív cellák és a döntési cellák között, itt összegződnek a bejövő jelek. A döntési cellák nyújtják a kimenetet. A mi példánkban a számítógépet a bináris osztályozásra tanítjuk be. Megnézi a program hogy az argumentumok száma kettő-e ha igen tovább fut ha nem értelemszerűen megáll. A terminál segítségével azt is megadja, hogyan is irassuk ki és futtassuk a programunkat. A megadott PNG fájlt beolvassa, amiben a perceptronunk segítségével eltároljuk a pixeleket. A for ciklusban előhívjuk a perceptront ami megadja nekünk a képünket.

Chapter 5

Helló, Mandelbrot!

5.1 A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/d0b50f83eb5a237eceba54c9d094e964

Tanulságok, tapasztalatok, magyarázat... Tutorált:Nagy László Mihály

Deklaráljuk a program elején a változókat, amik a kimeneti képünk méretét határozza meg plusz a vizsgált tartományt. Pixelenként megkapja a függvényből kapott paramétereket, amelyet egy tömbben tárol. A függvényben létrehozunk egy png kiterjesztésű fájlt, amibe elmenti a pixelre vonatkozó adatokat. Ezután kiírja a képfájlt.

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

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/264b04f23bd0132c3589c4eccb9ec03d

Hasonló az előzőhöz annyi különbséggel, hogy itt a c++ nyelv komplex osztályát használjuk ki.

5.3 Biomorfok

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

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

A biomorfok olyan alakzatok, amelyek ránézésre akár élő organizmusok is lehetnének, viszont nem muszáj természetes eredetűnek lennie az alakzatnak (magyarán, akár lehetnek számítógép által generáltak is).

Tanulságok, tapasztalatok, magyarázat... Biomorfok olyan alakzatok, amelyekre először élő organizmus is lehetne, de nem muszáj, ezáltall lehet mesterségesen létrehott is. Alapjának tekinthető a Mandelbrot-halmaz, ezek a hasonlóságok megtalálhatóak a programkódjukban is. Legfőbb eltérést az egyes pixelek színszámítás módjában lehet megtalálni.

5.4 A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/3b9c32e1535426bc598281a68197e890

Tanulságok, tapasztalatok, magyarázat... Ez a feladat (a fejezet) első feladatára épül, mivel a CUDA nem képes a device kódból függvényekre és osztályokra hivatkozni, amely csak a host-on találhatók. a "__device__" jelzi, hogy csak videókártyáról érhető el, hostról nem. A mandel függvény alkotja a mandelbrot-halmaz számolását pixelenként. A "__global__" jelzi, hogy kernelfüggvényről van szó. Itt határozza meg az aktuális szállat. Ez dönti el, hogy melyik számot kell éppen vizsgálni. A "cudamandel" függvény végzi a kernel meghívásához szükséges feladatokat. Pl:memóriatárolást a videókártyán A main függvény végzi el a cudamandel és a képfájl létrehozását.

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: Megoldás forrása: https://gist.github.com/AmidaHS/7c3855a3111b52b083fd3ab8dd4817ae

Tanulságok, tapasztalatok, magyarázat... A feladat megoldásához SFML-t használtam. A feladat hasonlít a fejezet második feladatához, így erre építek. A pixelek színét itt is a compute függvény adja meg. A mainben produkáljuk le a szükséges változókat valamint az objektumokat. A while-ban határozzuk meg az interakciókat, amit a felhasználó fog csinálni.

5.6 Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása: https://gist.github.com/AmidaHS/c32ba9b7b8d942fdb7f8772dcbb0d0f3

Tanulságok, tapasztalatok, magyarázat... Létrehozunk egy GUI-t. Megadjuk a paramétereket. Létrehozzuk az összes szükséges objektumot. A plotPoints rész végzi a bejáráshoz szükséges számításokat és magát a bejárást is. Az actionPerformed pedig a kölcsönhatást hajtja végbe a program és a felhasználó között.

Chapter 6

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:

C++: https://gist.github.com/AmidaHS/aae5212fd08103700c105bce57823c4d

Java: https://gist.github.com/AmidaHS/95fdfdd6b54cf07aa6006eacf51265c9

Először megadjuk a "Polargen" nwvű osztályt.Majd egy maximum 100-ig terjedő véletlenszám generátort, egyenlőre még nem tárolunk benne egy számot sem. A következő függvény generál két számot, egyiket eltárolja ami által logikai értéke megváltozik és hamis lesz.Másik számot visszaadja. Fontos szerepet játszik a nincsTárolt logikai változó, hiszen segítségével tudjuk eldönteni hogy páros vagy páratlan lépésben hívtuk meg a függvényt, amennyiben páratlan nem kell számolni csak az előző lépés másik számát adja meg. A main rész iratja ki az eredményt. Objektum orientált feladat, azért mondható könnyűnek, hiszen mi emberek mindent megtudunk objektumként határozni és ez itt sincs másként.

6.2 LZW

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

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/c62b6f09ee1a985e750d7086447b0487

Alkotunk egyfajta adatstruktúrát, ez a "Node", ezt 3 property-re osztjuk. "char c" , "left" és "Node*" . A "create_empty() függvény létrehoz egy új Node* pointert, a bal és jobb gyermekét beállítjuk nullára, ezután a függvény ezzel tér vissza. A "create_node(char val)" paraméterenként foglal helyett a memóriában,majd ezt eltárolja a "val" segítségével. A jobb és bal gyermekét nullára állítja ezután visszatér egy Node*-al. A

"void inorder" részben adjuk meg hogy ennek a fa bejárásnak inorderben történik a feldolgozási sorrendje. (1:bal, 2:gyökér, 3:jobb) A returnnel lépünk ki A depth adja meg a mélységet azaz ez minél mélyebben van annál jobbrább lesz. Az "int main" random generált számokat 2-vel oszt maradékosan így meghatározva a fának az értékeit, ezután kirajzolja.

6.3 Fabejárás

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

Megoldás videó:

Megoldás forrása: Az előző (6.2) feladat megoldásában megtalálható ennek a feladatnak a megoldása is.

A `preorder` eljárás annyiban különbözik az `inorder` eljárástól, hogy itt 1:gyökér, 2:bal oldal 3: jobb oldal.

A `postorder` eljárás annyiban különbözik az `inorder` eljárástól, hogy itt 1: bal oldal, 2:jobb oldal, 3: gyökér.

A `usage` kapcsoló segítségével adjuk meg a kimenetnek hogy in-/pre-/postordert használjunk.

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://gist.github.com/AmidaHS/fdc4244092afc792efb7954a78966941

Ez a megoldás nagyban épül az előző feladat (6.3) megoldására. A különbség, hogy a bináris fát kezelő függvényeket és eljárásokat a Binfa osztályba rendeztem, illetve a Binfa osztály privát részéve tettem a Node struktúrát. A binfa osztályon belül túlterheltem a balra bitshift operátort, amely mostmár a bináris fa építését látja el, ugyanazon elven, mint az előző feladatokban az `insert_tree` eljárás.

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://gist.github.com/AmidaHS/27d0179fd80cbc9c27bb7bf8332a984f

Hasonló az előző feladathoz egy kisebb átalakítással.A különbség abban mutatható ki, hogy mostmár a gyökérelemre is mutat egy mutató, emiatt a binfa konstruktornak adunk egy gyökérobjektumot. Még egy különbség, hogy a gyökérelemet kell átadni nem a referenciát a függvénynek vagy ahol a fát rajzolta ki, nem úgy mint ezelőtt.

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://gist.github.com/AmidaHS/c29b3b7d20aafda49e47fdc0d9e83df7

(6.4-hez hasonló) Egy olyan operátort hozunk létre ami ha nem tartalmazza a gyökeret akkor lemásolja önmagát.Ezt egy rekurzióval fejeződik be, ami minden ágat újra megalkot a másik gyökérre. A "rekurzioInditasa" függvény értelemszerűen elindítja a függvényt. Nullás gyermek esetén itt, egyes gyermek esetén azon is tovább fut. A "rekurzioazAgakon" átfut az összes ágon és az új csomópontokat is létrehozza. A "=" operátor segítségével az alap binFa-t átmásoljuk a binFa2-be.

Chapter 7

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/AmidaHS/p1

Tanulságok, tapasztalatok, magyarázat... A program látványosnak és érdekesnek mondható, hiszen a hangyák és feromonjaik általi kommunikáció terjedését akarja nekünk leszimulálni. A képet felosztjuk cellákra, majd megfigyelhetjük hogy milyen ütemben terjednek el és növekszik meg ezek a feromonok és, hogy hogyan alakítanak ki különböző "kitaposott ösvényeket".

7.2 Java életjáték

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

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/68ac0fde369d281d0da184d81298b341

Tanulságok, tapasztalatok, magyarázat... Ugyan úgy működik, mint a C++-os változata, kivétel, hogy itt kihasználjuk a Java adta lehetőségeket.

7.3 Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/9b3fc3182d34e2b34729cca5b04245a3

Tanulságok, tapasztalatok, magyarázat... A feladat SFML-el készült. Az 1. class rajzolja ki a hálót és ennek a részei jelképezik a sejteket. A 2. class tárolja a "sejtek" adatait:pl: hely, állapot. Ő felelős a kirajzolásukért is. Az update függvény vizsgyálja a sejteket a játék szabályainak megfelelően.

Szabályok

Egy sejt csak akkor marad aktív ha 2 vagy 3 aktív sejt van mellette. Ha egy sejt nem aktív csak akkor lesz aktív ha 3 sejt aktív mellette.

A killall célja, hogy megölje az összes sejtet ha editor modeban vagyunk. Ez tesztelés miatt található meg, hogy újraindítás nélkül lehessen tesztelni. A mainben jön létre az ablak és itt kapja meg a 10es fps határt a program. Ez után jönnek létre a változók. A fő ciklus, addig fut, amíg az ablak be nem záródik. A belső ciklus reagál a különböző eseményekre. A program akkor áll le ha az ablak bezáródik, vagy ha megnyomjuk a q gombot. Editorban egérrel állítható a "sejtek" inaktív és aktív állapota. A c megnyomása editorban minden sejtet inaktívvá vált. Ha nem vagyunk editorban elkezdi kirajzolni a játékot.

7.4 BrainB Benchmark

Megoldás videó:

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

Tanulságok, tapasztalatok, magyarázat... A "játékunk" azt figyeli, hogy mennyire tudunk a karakterünkre koncentrálni esetleg ha szem elől tévesztenénk akkor mennyi idő alatt találjuk meg. Ennek egyik legnagyobb előnye a különböző kompetitív játékokban jelent előnyt, ahol egyszerre több dolog történik és mindenhova figyelnünk kell, köztük a karakterünk aktuális pozícióját is. Legjobb példa ilyen helyzetre egy MOBA játéknak a csapatharc része, ahol egyszerre minimum 10 karakter jelenik meg és le kell tudni reagálni a kialakult helyzetet.

Chapter 8

Helló, Schwarzenegger!

8.1 Szoftmax Py MNIST

Python

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

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

Programunk egyik funkciója, hogy a kézzel rajzolt számokat felismerje. Először importájuk a tensorflow modult, aminek segítségével be tudjuk tölteni az MNIST adatbázisban tárolt adatokat. Az x és y_train valamint a x és y_test változókban tároljuk el az MNIST adatbázisból a képeket a tesztképeket és a címkéket. Ezután átkonvertáljuk az x-es tömbök értékeit float -ra, majd egységesítjük,hogy 0 és 1 közé essenek az értékek. Ezután kell 6 modult importálni ami segítségével jöhet létre a neutrális hálónk. Megalkotjuk a modelljét, majd "okossá" tesszük a hálót. Ezután pedig kiértékeljük a neurális háló "tudását" a teszt képekkel és címkékkel.

8.2 Mély MNIST

Python

Megoldás videó:

Megoldás forrása: https://gist.github.com/AmidaHS/aff80530f03b02bde17f6d67929effb0

A program elején importájuk a szükséges modulokat. Az importálások után létrehozunk négy függvényt, amelyek a programot hivatottak rövidíteni. A függvények után létrehozzuk a neurélis háló rétegeit. Ebben a programban 5 rétegű a neurális háló. A rétegek megadása után megadjuk, hogy milyen módon szeretnénk kiértékeltetni a bemenetet, majd megadjuk, hogy milyen algoritmussal tanítjuk fel a hálót. Ezután következik a tanítás, illetve minden századik iterációban leteszteljük a háló tudását.

8.3 Minecraft-MALMÖ

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

Megoldás forrása: https://gist.github.com/AmidaHS/80aad187e11d0d517933247b262295c1

Tanulságok, tapasztalatok, magyarázat... A Microsoft annak érdekében alkotta meg a Minecraft-MALMÖ projektüket, hogy megalkossanak egy MI-t a Minecrafton belül. A játék célja a karakterünk "Steve" folyamatos mozgásban tartása, jelen esetben 5 percen át. Ennek érdekében elemzi az MI az előtte lévő 12 blokkot és eldönti a legmegfelelőbb lépést, ami lehet: ugrás, elfordulás és más irányban való haladás. A példaprogramot felhasználtam a program megírásához. Legelőször importáljuk a MALMÖ API-t és a nélkülözhetetlen modulokat.A "run" azaz futtatás alatt található meg a küldetés pontos megadása, a "restart_minecraft" függvény pedig az újboli küldetés indításának lehetőségéért felel. A világ legenerálásának szabályait a "missionXML"-ben találhatod. Utána megadjuk az 5 perces idő limitet és a felbontást ami ezesetben 640x480. Két nagyon fontos while ciklust találunk. Az első segítségével addig várakozunk míg a felhasználó kész az irányítást fogadni. A második segítségével pedig karakterünket Stevet tudjuk vezérelni.Az ezutáni elágazásoknak köszönhetően karakterünk át tudja ugrani az akadályt vagy nem fog csak folyamatosan ugrálni, fordulni és "nézelődni" is képes lesz. Ezálltal tudja majd a program megvizsgálni a közelében lévő blokkokat. Ezt a vizsgálatot a "blocks" nevezetű 2D-s tömbre vetítve végzi el a program.

Chapter 9

Helló, Chaitin!

9.1 Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

https://gist.github.com/AmidaHS/d4605773c715520bbffe3dad3559ae01

https://gist.github.com/AmidaHS/3b0b3d4ea862f71877e44e3556312ecb

Iteratív: Definiáljuk a faktoriálásért felelőst "fact" függvényt.Majd az inputról kérünk egy számot. Erre a számra történik a fact függvény általl a faktoriális számítás.

Rekurzív: Hasonlóan az előzőhöz itt is létrehozunk egy függvényt, "factorial" néven, csak ez rekurzív módon számol, amelynek feladata a faktoriális számítások, amelynek matematikai hátterét a két if között láthatunk. Végül az ötös számra hívjuk a függvényünket.

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... Programunk elején definiáljuk az alap adatokat, mint például a szín palettát, magasságot,szélességet.Megalkotjuk a "image"-t és a "layer'-t, majd (step) lépésenként haladunk. Első lépésként az alap háttéret adjuk meg, majd fekete hátérre fehér színnel írást állítunk be amit összeillesztünk egy layer-nek. Utána csinálunk egy elmosást,majd a képünk színpalettájának megadunk egy kétoldalú határértéket. Újra elmosás következik.Színszerint kijelöléssel csinálunk egyfajta keretet. Új réteget adunk hozzá(ez lesz a main réteg). Létrehozunk egy kattintásra is működő színátmenetet. Megszabjuk a képünk tér hatásait. Legvégül pedig az elején lévő kód segítségével beállítjuk a görbe fémességét.

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... Definiálunk 3 függvényt az elején: először a "car" fv. segítségével több elemet is kiválaszthatunk, a második függvény a szöveg magasságát és szélességét határozza meg, míg a harmadik függvény az újonnan megalkotott képünknek adja meg a magasságát, szélességét és a színét. Létrehozunk egyfajta "foreground"-ot aminek színtén beállítjuk a színét. Utána a szövegen dolgozunk, majd jönnek a forgatások. A forgatások után kicsit csinosítunk a dolgokon,állítunk a méreteken, középre igazítunk és a színeket is állítjuk. Új középre igazított szöveg layer-t adunk hozzá, majd legvégül beállítjuk hogy ki legyen-e jelölve a kép vagy sem.

Chapter 10

Helló, Gutenberg!

10.1 Programozási alapfogalmak

[?]

II. heti előadás (11. oldal, az "1.2 Alapfogalmak" című rész):

Három különböző szintet különbötetünk meg a programozási nyelveknek:Az első a gépi nyelv, amely nyelvet a processzor nyelvének is nevezünk. A második az Assembly szintű nyelv, ilyenek a gépközeli nyelvek. A harmadik pedig a magas szintű programozási nyelvek például a Java vagy a C++. Mindig a processzornak megfelelő saját nyelvét kell alakítani, hogy az tudja a forráskódot értelmezni. Erre két módszer létezik: az egyik a fordítóprogramos, ahol a forrásszöveget lefordítja ezáltal válik futtathatóvá. És van az interpreteres vagy másképpen a soronként végigfutó, amely minden egyes sorban külön végzi el az utasításokat. Minden programozási nyelvnek megvannak a maga sajátosságai, a hivatkozási nyelvükbe van levezetve ezek a szabályok. Léteznek implementációk is, nem kompatibilisek egymással. Több is megtalálható egy operációs rendszeren belül is akár, de ezek sem lesz egymással kompatibilisek. Sokkal egyszerűbb IDE-ket használni.

III. heti előadás (28. oldal, a "2.4. Adattípusok" című rész):

Az adattípusoknak van saját neve amikkel azonosíthatóak típus szinten.Különböző programozási nyelvek különböző szinten használják ezeket az adattípusokat, vannak amelyek sokat és olyan is amely szinte soha vagy valóban soha. A programozó maga is megadhatja a változók típusát. Ilyen nyelvekre lehet példa a C++ vagy a Java.Amelyek nem használnak típusokat azokban automatikusan dől el hogy milyen a változó.Ilyen az R és a Python is. Létezik összetett és egyszerű adattípus,az egyszerűeket nem lehet tovább bontani, míg az összetetteket igen.(ilyen lehet pl a struktúrák vagy a felhasználó általi változók)

III. heti előadás (34. oldal, a "2.5. A nevesített konstans" című rész):

Két fő feladatát különböztetjük meg: 1: definiálásnál meg kell változtatni az értékét. 2:olyan név, amely jelzi az értéket és a típust is. Ennek a definiálása nem maradhat el!

III. heti előadás (35. oldal a "2.6. A változó" című rész):

Négy részt különböztetünk meg egy változónál: név, attribútumok, cím és érték. Név: azonosító, névhez rendeljük. A típusa határozza meg főleg, amely a felvehető értékét is meghatározza.Ezeket deklarációból kapja meg.Több fajta deklarációt is ismerünk: 1:implicit 2:explicit 3:automatikus A cím határozza meg a betöltött helyét, ennek három változata van: 1:dinamikus 2:statikus 3:programozó által vezérelt Érték meghatározásnak módjai: 1:kezdőérték adás 2:értékadó utasítás által

III. heti előadás (39. oldal, az "2.7. Alapelemek az egyes nyelvekben" című rész):

Aritmetikai típusról beszélünk ha egyszerű típusú és származtatottról ha összetett tipusú. Belső kódokból épül fel a karakter típusa. False = int 0 a true = int 1 A struktúra fix, logikai típus nincs. Különböző elemeknek lehet ugyanaz az értéke.A void tartománya üres, így nincs művelete se.

IV. heti előadás (46. oldal, az "3. Kifejezések" című rész):

A szintaktikai eszközöknek két féléjét különböztetjük meg: értéke és tipusa. 3 különböző dormális összetevőről beszélhetünk: kerek zárójel, operandus és eporátor. 3 alakban lehet a kifejezéseket felírni, (operátor helyzetétől függően) sorrendben: legelől PREfix, INfix alak középen és POSTfix alak leghátul. Végrehajtási sorrend is lehet többféle kép, méghozzá: balról-jobbra vagy jobbról-balra.

V. heti előadás (56. oldal, az "4. Utasítások" című rész):

Két nagy csoportot különböztetünk meg :1:deklarációs utasítás 2:végrehajtó utasítás. az "1"-es fordítóprogramhoz szól,míg "2" generálja a tárgykódot a fordító programhoz. Végrehajtásos utasításba tartoznak: A:Értékadó utasítás , B:Üres utasítás, C:Ugró utasítás, D: Elágaztató utasítás, E: Ciklusszervező utasítás F: Hívó utasítás, G: Vezérlésátadó utasítás, H: I/O utasítások, I: Egyéb utasítások.

VII. heti előadás (78-84. oldal):

Ebben a részben a paraméter átadás több módjával ismerkedünk meg. Érték szerint: formális paraméter értékének megkapja az aktuális értéket.Ezt nem lehet megváltoztatni, amennyiben ezt használjuk. Cím szerinti: Megkapja az aktuális címet a paraméter,ilyenkor meglehet változtatni használat ellenére is. Eredmény szerinti:Szintén az aktuális címet kapja meg, de nem használja azt, csak majd a végén betölti ezeket az adatokat. Érték-eredmény szerint: A cím másolódik, használja az adatot, majd be is másolja azt.

VIII. heti előadás (82. oldal, a "A blokk" című rész):

A VI. heti előadáson elhangzott. Ott található legfőképpen.

VIII. heti előadás (83. oldal, a "Hatáskör" című rész):

Röviden: program azon része amit jelöl. Név hatásköre a programegység.Az itt megadot név lesz a lokális név.Szabad név ha nem deklaráljuk csak hivatkozunk egy egységben. Hatáskör kezeléskor meghatározzuk a név hatáskörét. Két fajta van:1:statikus 2:dinamikus. Mindig befelé terjed. Globális névnek nevezzük ha nem lokális de látható lokálisan. Ezek relatív fogalmaknak tekinthetők. 1: összes név hatásköre egyértelmű 2:a futtatás alatt is változhat, akár minden alkalommal más. Statikus nyelvet használnak az eljárásorientált nyelvek.

VIII. heti előadás (98. oldal, a "Absztrakt adattípus" című rész):

Adattípus, mely információ rejtéssel és/vagy bezárással foglalkozik.Ehhez az interface-én keresztül juthatunk el, ami által minden hibalehetőség ki van zárva. Utóbbi időben teret nyert magának és fontos szerepe lett.

VIII. heti előadás (121. oldal, a "Generikus programozás" című rész):

Bármely programozási nyelvbe behelyettesíthető. Megadunk egy paraméterezhető szövegmintát, amelyet a fordító kezel, bármennyi szöveg előállítható a mintaszöveg paramétereinek köszönhetően.

IX. heti előadás (134. oldal, az "Input/Output" című rész):

Az I/O a perifériákkal való kommunikációt és kapcsolattarást végzi, a létrejött adatátvitelnek két módja van:Folyamatos, ebben az esetben eltérő ábrázolás és bináris. Kialakult eszközrendszernek tekinthetőa dormátumos módú adatátvitel, a listázott módú és a szerkeztett módú adatátvitel. I/O-ban az állomány van a kozponti helyen. Funkció szerint 3 fajtát azonosítunk: input, input-output és output állományt. Állományok használásához szükséges a deklaráció,összerendelés,állomány megnyitás, feldolgozás végül pedig a lezárás.

10.2 Magas szintű programozási nyelvek 2 by Juhász István (Pici könyv 2)

XI. heti előadás (38. oldal, a "Kivételkezelés a Javaban" című rész):

Alapvető eszköz. kivétel objektum jön létre, ha egy olyan eset áll fenn amikor a java működése alatt speciális esemény történik.Ekkor ez a Javának a virtuális gép hatáskörébe fog tartozni. A JVM-nek kell kezelni a helyzetet.Akkor találja meg ha a talált típus megegyezik vagy ha őse a kivételnek a talált típus. Ez a blokk bárhol elhelyezhető és egymásba is szedhető.

10.3 Programozás bevezetés

[KERNIGHANRITCHIE]

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

V.heti előadás (Függelékből az Utasítások című fejezet):

A különböző utasítások a betáplálás sorrendjében történik meg. Több csoportra osztható:címkézett utasítás, goto utasítás. Címkének nevezzük az azonosítás nélküli deklarált azonosítót.PL:utasítások nagy része. Kiválasztott utasítás:Mindig a végrahajtási sorrend egyike lesz a választott. Iterációs utasítás:ciklust határoz mge. Összetett utasítás: Neve is utalja, egyszerre több utasítást egyszerre hajtja végre, szükséges a forításhoz. Vezérlésátadó utasítás: Neve árulkodó, vezérlés átadására alkalmazzák.

10.4 Programozás

[BMECPP]

V.heti előadás (1.-16.):

A c++ nyelv a C továbbfejlesztett változata, amely sokkal kényelmesebb és biztonságosabb is. Míg C nyelvben a függvény üres paraméterlistával definiálva tetszőleges számmal hívható, addig C++-ban sokkal egyszerűbben egy void megadása. A bool változó true vagy false értéket vehet fel, míg a wchar_t beépített típusú. C++ lehet változót deklarálni, sokkal átláthatóbb így.Lehet két azonos nevű függvényt létrehozni C++ban hogyha az argumentumukban van eltérés. És már a függvényeknek alap arg. értéket is lehet adni.

VI. heti előadás (17-58.):

C++ osztályokról szól a rész. Egy objektum orientált prog.nyelv lényege, hogy leegyszerűsítse az emberek számára a programozást az objectek és classok bevezetésével.Így sokkal átláthatóbban és egy egységben lesznek az összetartozók, ezeket nevezzük tagfüggvény és adattagoknak. Privatel és Protecdel által létrejön az adatrejtés.Logikusan célja külső helyről ne lehessen elérni.a konstruktor feladata a lefusson az object, a destruktor feladata pedig hogy miután lefutott a megsemmisült object után dinamikus tag felszabadítása. Nem tartozik az osztály tagjai közé mégis eléri a private tagokat a friend osztály és a friend függvények. Tagváltozás történhet: -konstruktoron belül, -tagfüggvényel, -külső függvényel. Az osztályokban szerepelhet define is, amely lehet: struktúra, osztály és enumeráció is.

VII. heti előadás (93-96.):

Operátorok és túlterhelésük: A műveletek az argumentumokon hajtják végre az operátorok.Ennek a visszaadott értékével dolgozunk. A c++-os operátorokat túl tudjuk terhelni("110%-ot ad bele",Több célra is tudunk egy operátort használni.), de új operátort nem tudunk alkotni.

IX. heti előadás (73.-90.):

C-ben 3 File állományleíró tipúst különböztetünk meg:stdin a bemenet, ami csak olvasásra, stdout ami a kimenetet adja csak írásra és a stderr ami a hibás kimenetért felel szintén csak írásra. Míg C++-ban ott vannak az objektumok és az operátorok be- és kiírásra is egyaránt.Beolvasásnál a cintől elfele, kiíratásnál pedig a cout irányába történik az adat áramlása. A cinnel vigyázni kell mert csak addig hajtja végre a feladatát ameddig a várt típust kapja.

(93-96)

Operátorok és túlterhelésük.

C nyelvben mellékhatásnak nevezzük, mikor az operátor az arg értékét módosítja, mivel alapból a visszatérési érték szokta megadni, így az argumentum sem változik. Zárójelekkel tudjuk az operátorok kiértékelésének a sorrendjén változtatni. És ez esetben is mint szinte minden másban, a C++ sokkal kelzelhetőbb és felhasználó barátabb mint őse a C.

C++-ban a függvények képesek bizonyos mellékhatásokra is, míg ez a C nyelvre nem mondható el. Függvényhez hasonlóan az operátorok alkalmazásának definiálását is a függvényszintaxis segítségével megoldhatjuk. Mind a függvénynevek mind az operátornevek túlterhelhetőek, hiszen a függvényeknek ez egy egyedi tulajdonsága és az operátorok is egyedi függvények.

XI. heti előadás (187.-197.):

Ha valahol megszakítás van, akkor a hibakezelő "ágon" folytódik a kivételkezelésnek köszönhetően. Ide nem csak a hibás eseteket értendőek, hanem a kivételes eseteket is.

Egy példa a kivételkezelésre:

```
#include <iostream>
usning namespace std;
int main()
{
  try
  {
    double d;
    cout << "Enter a nonzero number: ";
    cin >> d;
    if(d == 0)
    {
     throw "The number can no be zero.";
    }
    cout << "The reciprocal is: " << 1/d << endl;
  }
  catch (const char* exc)
  {
    cout << "Error! The error text is: " << exc << endl;
  }
  cout << "Done." << endl;
}
```

Könyvbeli példa: Kérünk egy nem nulla double típusú számot,ez lesz a d változónk, majd iffel leellenőrizzük hogy tényleg nem-e nulla. Ha nulla akkor a "throw" által lehetséges hibaként kezeljük, amennyiben nem nulla, megmondjuk a reciprokát, amit ki is írunk. "catch"-ben a hiba esetén kiirandót írjuk ki majd a vérére a done-nal konstatáljuk a végét. A "catch" részben elkapjuk a "throw" által eldobott hibát, és kiírjuk, az "if"-ben megadott mondatot. Mindkét megoldás végén a program kiírja, hogy "Done.".

```
A kimenet, ha a felhasználó nem nullát ad meg:

    Enter a nonzero number: 2
    The reciprocal is: 0,5
    Done
```

```
A kimenet, ha a felhasználó nullát ad meg:

    Enter a nonzero number: 0
    Error! The error text is: The number can not be zero.
    Done.
```

A "try-catch" blokk egymásba ágyazható. Alkalmunk van néhány közeli értékű kivételt alacsony szinten elkapni valamint használni is. Ha nem használjuk a throw paramétert akkor a kivételt megváltoztatjuk. Ezen folyamat alatt, míg egy kivétel nélkülit nem kapunk a helyi változók rendre felszabadul. Ezt nevezzük a folyamat visszacsévélésének.

```
A verem visszacsévélése:
    int main()
    {
     try
     {
       f1();
     }
     catch(const char* errorText)
     {
       cerr << errotext << endl;
     }
    }
    void f1()
    {
      Fifo fifo; //a fifo egy általunk megírt osztály
      f2();
      ...
    }
    void f2()
    {
      int i = 1;
      throw "error1";
    }
```

Magyarázás:

F2 kivételt jelez, majd az "i" helyi változó felszabadul. Ezután az f1-ben lévő "Fifo fifo" is felszabadul, mert a destruktor által meghívódik. Lefut a catch.

Part III

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.

Chapter 11

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...

Part IV

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.