Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Boros, István Tamás	2019. október 24.	

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
0.1.0	2019-02-27	Első fejezet kész	Boros István Tamás
0.2.0	2019-03-03	Második fejezet kész	Boros István Tamás
0.3.0	2019-03-10	Harmadik fejezet kész	Boros István Tamás
0.4.0	2019-03-16	Negyedik fejezet kész.	Boros István Tamás

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
0.5.0	2019-03-24	Ötödik fejezet kész	Boros István Tamás
0.6.0	2019-03-30	Hatodik fejezet kész	Boros István Tamás
0.7.0	2019-04-07	Hetedik fejezet kész.	Boros István Tamás
0.8.0	2019-04-13	Nyolcadik fejezet kész.	Boros István Tamás
0.9.0	2019-04-19	Kilencedik fejezet kész	Boros István Tamás
0.11.0	2019-04-28	Tizenegyedik fejezet kész.	Boros István Tamás
0.11.1	2019-05-05	Javítások és módosítások.	Boros István Tamás
1.0.0	2019-05-09	Kiadás	Boros István Tamás

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

Táblázatok jegyzéke

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

```
Végtelen ciklus 0%

#include <unistd.h>
int main()
{
for(;;) sleep(1);
return 0;
}
```

```
Végtelen ciklus egy magra

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

```
Végtelen ciklus minden magra

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

```
    }
    return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat... : A feladat lényege, hogy egy olyan programot írjunk, amely folyanatosan fut és 0%-ban, 100%-ban lefoglal egyet és 100%-ban lefoglal minden processzor magot. Ezt egy olyan ciklusba kell foglalni melynek feltétele mindig igaz lesz, ezáltal végtelen köröket fog futni a program. A for ciklus három kifejezését üresen hagyjuk, mivel sem deklaráció, sem feltétel, sem ciklus végi művelet nem lesz. A ciklus így a végtelenségig fog futni. A minden magot lefoglaló programhoz szükséges az omp.h header és a main-be a #pragma omp parallel sor. A fordításhoz -fopenmp kapcsoót kell használni (ha többször írjuk be a kapcsolót a fordításhoz egyel több magot fog lefoglalni).

Ebben a feladatban segítettem Nyitrai Dávidot

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...: Mivel minden esetben lefagy a program, ezért nem létezik olyan program, ami el tudja dönteni egy programról, hogy van-e benne vételen ciklus. A valódi kérdés az a végtelen ciklus detektálásában van, amit Turing 1936-ban állapított meg, hogy nem lehetséges.

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/BOPI98/BHAX_feladatok/blob/master/3csere.c

Tanulságok, tapasztalatok, magyarázat... : Ebben a programban alapvető matematikai műveletekre és egy kiiratási függvényre lesz szükség, ezért egy stdio headert használunk. A fő függvényben először is deklarálunk két változót, ami ebben az esetben két egész szám. Ezt a két számot printf() függvénnyel kiiratjuk. A printf függvényben az idézőjelen belül beírjuk a megjelenítendő szöveget. Ha változókat akarunk elhelyezni a kiiratásban akkor azt a változó típusával kell helyettesíteni (jelen esetben %i, az integer típus miatt) és a változónevet rendre vesszővel elválasztva kell felsorolni az idézőjel után. Az idézőjelen belüli utolsó két karakter (\n) a sortörést jelzi, amely nem jelenik meg a végső kiiratásban. Ezután a csere következik. Mivel segédváltozó nélkül kell dolgoznunk, ezért összeadással és kivonással fog a csere megtörténni. A csere után újabb kiiratást teszünk, majd return 0 visszatérő értékkel jelezzük hogy sikeresen lefutott a program.

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: iffel: https://github.com/BOPI98/BHAX_feladatok/blob/master/4labda.c

if nélküli: https://github.com/BOPI98/BHAX_feladatok/blob/master/noif.c

Tanulságok, tapasztalatok, magyarázat... : A programban létre kell hozni egy screen-t, és ezen fogjuk elhelyezni a pattogó labdát. Deklarálunk egy növekvő értéket és x, y koordinátákat. Végtelen ciklusban léptetjük a labdát és ha a screen szélére ér, akkor negatív növekvő értékre váltunk. Minden alkalommal clear-elni és usleep-elni kell a programot a ciklusban.

Ebben a feladatban segítettem Nyitrai Dávidot

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/BOPI98/BHAX_feladatok/blob/master/5szohossz.c

https://github.com/BOPI98/BHAX_feladatok/blob/master/bogomips.c

Tanulságok, tapasztalatok, magyarázat... : Ez a program egy int váltózónak számolja meg, hogy hány bitből áll. Az elején deklarálunk egy int változót pl:a és deklarálunk egy megszámlálás tételéhez szükséges egész számot. A while ciklus feltételében addig bitshiftteljük a számot, amíg teljesen le nem nullázódik. A ciklus végén mindig növeljük a számlálót és ez adja meg a szó hosszát. A bogomips programban egy delay függvényre lesz szükség ami bevesz egy loop számot és ennyiszer lefuttat egy üres ciklust. Majd a main függvényben deklaráljuk a loop/sec változót és a tick-et. Egy while ciklusban addig fut a ciklusmag amíg bitshifttel 0-t nem kapunk a loop/sec-ből. A ciklusmagban kiszámítjuk az órajel különbséget a delay

függvény meghívása után. Majd kifejezzük és kiiratjuk a loop_per_sec értékét. Ha hamis érték tér vissza, failed-et ír ki a program.

Ebben a feladatban segítettem Nyitrai Dávidot

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/BOPI98/BHAX_feladatok/blob/master/pagerank.c

Tanulságok, tapasztalatok, magyarázat... :A program célja, hogy négy page-et sorrendbe rendezzen egy pagerank algoritmus alapján. Minden page mutat másokra, amit egy mátrixba rendezve tudunk eltárolni. Ezután lefuttatjuk a megfelelő algoritmust az elemekre és így megkapjuk az értékeket.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

A Brun tétel azt demonstrálja, hogy az ikerprímek (két egymás utáni ikerprím különbsége 2) szummája egy számhoz haladnak. A tételt Viggo Brun bizonyította 1919-ben és szitaeljáráshoz használják. Irracionális Brun-konstans, akkor lehet, ha végtelen sok ikerprím van. Létezik egy brun konstans prímnégyesekre is ami két ikerprímpárból áll. A programban a megadott értékig legenerálja a prímeket, majd az egymásra rákövetkező prímek különbségét változóba tárolja. Ha a különbség kettő, akkor ikerprímként tároljuk el, majd egy másik tömbben az ikerprímek második tagját tároljuk. Végül a két tag recoprokértékének összegeit összeadjuk.

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 Monty Hall probléma arról szól, hogy ha három lehetőség közül csak az egyik lesz kedvező, majd választunk egyet és a fentmaradó lehetőségek közül megmondjuk, hogy az egyik nem a kedvező lehetőség, akkor érdemes e megváltoztatni a választásunkat vagy nem. Mivel az eredeti fennállásból adódóan 33.33% esélyünk van és minden másnak 66.66% esélye van így, ha van lehetőség a választásra nagyobb valószínűsége van annak, hogy a kedvezőt választjuk ha cserélünk. A szimuláció abból áll, hogy több kísérletet teszünk a Monty Hall problémára. Mivel az esetek kimenete nem állandó így eredményt csak a statisztikán keresztül láthatjuk.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/un%C3%A1ris.png

Tanulságok, tapasztalatok, magyarázat... :A számrendszer a legegyszerűbb számrendszer és természetes számokra használják. A gép mindig annyi darab egyest fog visszaadni, mint amennyi a természetes szám értéke. A számok könnyebb kiolvasása miatt jött létre ez a számrendszer. A számolás ötösével történik az ujjak száma miatt. A képen látható, hogy természetes számokból hogyan keletkezik az unáris szám.

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

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

Megoldás forrása: Első

```
Szabály:

  S->aBSc
  S->abs
  Ba->aB
  Bb->bb


  Levezetés:

  S -> aBSc -> aBaBScc -> aBaBabccc -> aaBBabccc -> aaBaBbccc -> aaaBBbccc ↩
     -> aaaBbbccc -> aaabbbccc
```

Második

```
Szabály:

S -> abc
```

```
S -> aXbc
Xb -> bX
Xc -> Ybcc
bY -> Yb
aY -> aaX
aY -> aa


Levezetés:

S -> aXbc -> abXc -> abYbcc -> aYbbcc -> aaXbbcc -> aabXbcc -> aabbXcc ->  ↩
   aabbYbccc -> aabYbbccc -> aaYbbbccc -> aaabbbccc
```

Tanulságok, tapasztalatok, magyarázat... : A Chomsky által megalkotott nyelvnek három fajtája van, környezetfüggetlen, genetatív és szabályos. A környezetfüggő nyelvnél a szabályok szerint kell végrehajtani a fordítást. A grammatika azt adja meg, hogy hogyan hozzunk létre egy mondatot az adott nyelven. A grammatikának négy alkotóeleme van, a terminális szimbólumok, a nem terminális jelek, kezdőszimbólum, és helyettesítési szabályok.

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/BOPI98/BHAX_feladatok/blob/master/hivat.c

```
#include <stdio.h>
void main ()
{
long double a=100000000000000000; //a C89-es szabvánnyal nem fut le;
printf("%Lf\n",a);
}
```

Tanulságok, tapasztalatok, magyarázat...: C-ben egy parancssor mindig pontosvesszővel végződik. Ha ciklusokat használunk vagy függvényeket definiálunk akkor {} kapcsos zárójeleket haszlálhatunk az azon belüli parancssorok elkülönítésére, de ezzől még ez is parancssor.

A kódcsipetben a fő függvény void-ból áll és azon belül long double-t irat ki. Ezek a C99-es szabványba kerültek bele, így nem futtatható a C89-esben.

3.4. Saját lexikális elemző

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

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/lex.c

Tanulságok, tapasztalatok, magyarázat... : Az első %{}% részben hozzáadjuk a könyvtárakat és deklaráljuk a megszámláló változót. A második részben a bemenetből megszámoljuk, hogy hány szám érkezett be. ha betű vagy írásjel érkezik azt a program ignorálja. a mainben lefut a függvény és kiiratjuk hány szám érkezett be.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/leet.c

Tanulságok, tapasztalatok, magyarázat... Az első lex részben elhelyezzük az include függvénykönyvtárakat, deklarálunk egy változót majd definiáljuk a karakter struktúrát, a cipherbe elhelyezzük a karakterekhez párosítandó hasonló karaktereket. A második részben megkeresi a beolvasott karaktereket a tömbben és kicseréli egy random hasonló karakterre. Az utolsó rész a main függvény ahol meghívjuk a függvényt és lefut a leet program.

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:

```
#include <stdio.h>
#include <signal.h>
void jelkezelo()
{
printf(" Valami\n");
}
int main()
{
while(1)
{
if(signal(SIGINT, jelkezelo)==SIG_IGN)
    signal(SIGINT, SIG_IGN);
}

}
```

Tanulságok, tapasztalatok, magyarázat... :A program egy végtelen ciklusban teszteli a bemenetet és elkapja a megszakításokat. Miután elkapott egy magszakítást meghívja a jelkezelo függvényt.

i. Ez a kód signal függvénnyel kap el leállító parancsokat és ami lefuttatja a jelkezelo függvényt, ha az első sorban lévő signal függvény nem egyenlő a SIG_IGN-el.

ii.,iii.: Ez a két kód ugyan olyan for ciklus, melyek 5-ször fognak végig menni a ciklusmagon. Az i változót korábban deklarálni kell különben nem fog lefutni.

iv.: Ez a programkód igényel korábbi deklarációkat (pl:i, és a tomb[i] tömb cellaszámmal, mivel ha túl lépi a ciklusban a maximumot, akkor lehetséges, hogy nem fut le a program.

v.: Ez a kódcsipet egy hasonló for ciklus az előzőkhöz annyi különbséggel, hogy a feltételben két argumentum van. A második, két pointer rákövetkezőjét teszi egyenlővé. Az előzőben látott tömb probléma most is fennállhat a d pointernél.

vi.,vii.,viii.: Ez a három csipet kiirat két decimális számot, melyeket függvényből kapunk. Az argumentumokban van referencia, rákövetkező és sima változó.

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

Van olyan y, amelynél minden x kisebb, és y prím.

Van olyan y, amely nagyobb minden x-nél, és y prím és SSy prím.

Ha minden x prím, akkor van olyan y ami nagyobb minden x-nél.

Ha van olyan y kisebb, mint minden x, akkor x nem prím.

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

Tanulságok, tapasztalatok, magyarázat...: A latex-ben lefordított logikai kifejezések kvantált formulákat adnak vissza.

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;
  ```

  egyszerű egész deklarálása
- ```
  int *b = &a;
  ```

  Egész mutató egy a referenciára.
- ```
  int &r = a;
  ```

  egész referencia deklarálása, ami gcc-vel nem működik ezért c++-ban kell fordítani.
- ```
  int c[5];
  ```

  egész tömb.
- ```
  int (&tr)[5] = c;
  ```

  referenciatömb deklarálás
- ```
  int *d[5];
  ```

  pointer tömb
- ```
  int *h ();
  ```

  mutatót visszaadó függvény
- ```
  int *(*l) ();
  ```

  a fenti függvényre mutató függvény
- ```
  int (*v (int c)) (int a, int b)
  ```

  olyan függvény amibe két egész kerül és egy függvényre mutató mutató egészet ad vissza.
- ```
  int (*(*z) (int)) (int, int);
  ```

  a fenti függvényre mutató mutató ami egészet ad vissza.

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/dek.c https://github.com/BOPI98/-BHAX_feladatok/blob/master/dek2.c https://github.com/BOPI98/BHAX_feladatok/blob/master/dek3.c https://gi BOPI98/BHAX_feladatok/blob/master/dek4.c

Tanulságok, tapasztalatok, magyarázat...: Az első kettő sor könnyen lefut, a harmadik viszont nem tud, mert referenciát nem tudunk c-ben deklarálni. A negyedik egy sima egész tömb. Az ötödikben hasonlóan a hármashoz referencia miatt nem fut le a kód. A hatodik, hetedik vagyis egy egészre mutató tömb és egy egészre mutató mutatót visszaadó függvény gond nélkül lefut. A nyolcadik egészre mutatóra mutatót visszaadó függvényre mutató mutató is lefut. A kilencedik tizedik csipetben függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényt bevezettem egy programba.

Ebben a feladatban segítettem Nyitrai Dávidnak

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás videó: https://youtu.be/1MRTuKwRsB0

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/haromszog.c

Tanulságok, tapasztalatok, magyarázat... : Ez a program egy háromszög alakú kiiratást fog végrahajtani, amely úgy épül fel, hogy a program double változókat allokál a memórában és egy ciklus segítségével soronként tudjuk növelni az allokációk méretét. Malloc függvény fog mutatni az adott sor méretére. A program végén free függvénnyel lefuttatva megkapjuk, hogy hány lefoglalt allokáció van a memóriában.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:https://github.com/BOPI98/BHAX_feladatok/blob/master/exort.c

Tanulságok, tapasztalatok, magyarázat...: A programban define-oljuk a buffer méretet, és a max kulcsméretet, majd a main függvényben kulcsot és buffert deklarálunk. Utána while ciklusban a beolvasott adattal lefuttatjuk a for ciklusban lévő titkosítást. A programban ellenőrzni az atgumentum mennyiséget, hogy kettő-e. Ez a helytelen input megelőzése miatt van. Hiba esetén errorral kilép a program. Ha sikeres akkor megtörténik a beolvasás és lefut az exor. Ha ezzel végzett, akkor helyes lefutást jelezve kilép a program.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/exor.java

Tanulságok, tapasztalatok, magyarázat... : A program egy tiszta.txt fájlból olvassa be a titkosítandó adatot, utána bekér egy kódot, aminek egyeznie kell hogy lefusson a titkosítás, majd a titkosító ciklusok után visszairatja a fájlban a fordított szöveget.

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/BOPI98/BHAX_feladatok/blob/master/exor.c

Tanulságok, tapasztalatok, magyarázat...: A programhoz stdio, unistd, stdlib és string header-ekre lesz szükség. Az első része egy átlag szóhossz számító double függvény, ahol paraméterként meg kell adni egy konstans karaktermutatót ls egy integert. Ez a függvény a szóközök megszámlálásával számítja ki az átlagot. A második függvény azt vizsgálja, hogy az átlag szóhossz 6 és 8 között van-e és, hogy van-e benne "hogy, nem, az, ha" szó. Ezt követi egy void függvény, amihez egy konstans karaktertömb kell a kulcshoz, kell ennek a mérete, maga a titkos szöveg, a mérete, és a buffer karakterpointerként. A függvény kiszámítja a buffereket, majd a következő cikluskörhöz a kulcsindexet. Utána jön az exor törés függvény amihez paraméterként ismét kell egy kulcs, kulcsméret, titkos szöveg és a mérete. A függvényben, ha a buffer NULL akkor failed-et ír és újra lefut az exor. Majd ha a buffer lefut igazként a tiszt a függvényben akkor kiiratja a kulcsot és a Tiszta szöveget. A függvény végén free függvénnyel kiiratja a szaban és foglalt memória allokációkat. A main függvényben deklarálunk és végrahajtjuk a beolvasást, majd a kulcs megadással lefuttatjuk az exortörőt.

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...: Ez a program létrehoz egy neurális hálót, ami alapvető logikai műveleteket hajt végre, mint például or, and és exor. A műveletek naggyából egy mintára épül, ahol megadjuk a bemenet alapján a neuralnet függvénnyel, hogy milyen eredményhez kell közelítenie a thresholdon belül. A prot függvénnyel kirajzoljuk az eredményt.

A feladatban segített: Nyitrai Dávid

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/perceptron.cpp

Tanulságok, tapasztalatok, magyarázat...: A program elején egy if-fel teszteli hogy az 2-e az argumentumszám és ha nem, akkor hibát dob. Ezután beolvassuk a fájlt és neurális hálóval letrehozunk egy perceptront. Majd egy tömböt hoz létre annyi karakterrel ahány pixeles a kép soronként. Ebbe a tömbbe a kép piros értékét belemásolja, majd meghívja a preceptront és return 0-val kilép.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/mandelbrot.cpp

Tanulság: A mandelbrot halmaz egy komplex számokból álló rekurzív sorozat. A megjelenítés úgy történik, hogy egy algoritmus alapján kiszámított pontokat a komlex számsíkon ábrázoljuk. Mivel rekurzív függvényről van szó ezért a végtelenségig részletes lesz a kép. A programhoz png++ headert kell használni a képek elkészítéséhet. Az első sorokban definiálunk méreteket a kimeneti képről és az algoritmushoz szükséges értékeket is itt adjuk meg. A main előtt még létrehozzuk a generáló függvényt és egy komplex struktúrát. A generáló függvény kiszámítja minden pixel értékét rgb_pixel függvénnyel. A mainben létrehozzuk a kép tömbjét, deklaráljuk az algoritmushoz szükséges változókat, majd ciklussal minden pixel értékét kiszámítjuk, és meghívjuk ezekre az értékekre a generáló függvényt.

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

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/mandelbrot_komplex.cpp

Ez a feladat ugyan az mint az előző csak itt nem struktúrát hozunk létre a komplexhez hanem headerből hívjuk meg. Az algoritmus nem változott és a generáló függvény is ugyanúgy működik, mint az előzőben. A síkbeli alakzat alapvetően egy egyszerű algebrai összefüggést rajzol ki. Az alakzat semmiben sem különbözik az euklideszi síktól, egyedül a pontok meghatározása másabb.

5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat... : A program ugyanaz, mint az előzők, annyi különbséggel hogy ítt a program indításával visszük be az adatokat. A képet ismét az rgb_pixel függvénnyel készítjük, az

algoritmus lefutása után write függvénnyel az első argumentum nevén elmenti a png-t. A program egy másik mintát állít elő mint a mandelbrot halmazban látható, de a tematika itt is hasonló. Ennél a programnál bár a kép lehet végtelen részletes, mégsem keletkezik új minta, ahogy a mandelbrotnál.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/mandel_cuda.cpp

A CUDA egy Nvidia által létrehozott eljárási módszer, ami párhuzamosan számít ki adatokat. Ez a megoldás általában mátrixoknál hasznos és kép, vagy grafikus számításoknál használandó. A CUDA megvalósítás gyorsabb lefutási sebességet eredményez a pixelek kiszámolásakor. A CUDA előnyeként szolgál, hogy a rendezés sokkal gyosrabban történik és a gpu-k között gyorsabb letöltés és visszajelzés van.

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://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/mandel_nagyito/-

Magyarázat: A program ugan úgy épül fel mint az előzők. A nagyító úgy működik, hogy egy pontra kattintva kiszámítja a ráközelített kép adatait és ezekkel az adatokkal újra elvégzi a számítást. Mivel minden alkalomman újragenerálja a képet ezért a végtelenségig fog menni a kép. A Qt programmal létrehozott grafikus felület tulajdonképpen annyiból áll, hogy egérkattintás inputtal közelít a képre, vagyis generálja le egy közelebbi változatát. Ezzel a funkcióval akár a végtelenségig is generálhatjuk a mégközelebbi képeket.

5.6. Mandelbrot nagyító és utazó Java nyelven

forrás: https://github.com/BOPI98/BHAX_feladatok/blob/master/nagyitoutazo.java

magyarázat: A programban egy guival fog működni a nagyító, amit egy konstruktor osztályban hozunk létre. Ennek paramétereit megadjuk méret és annak méretezésével, majd egy kontroll objektumot hozunk létre. A plotPoints-al kiszámoljuk a halmaz elemeinek értékeit, a actionPerformed-del pedig a felhasználói interakciót vezethetünk be.

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/BOPI98/BHAX_feladatok/blob/master/polargen.cpp

https://github.com/BOPI98/BHAX_feladatok/blob/master/polargen.h

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 program egy headerben létrehoz egy osztályt, ami tartalmaz egy publikus Polargen részt, ami egy boole-anból áll majd deklarálva van a random generátor. Ezután jön a privát bool deklaráció és a tárolt értéke doubleben. A cpp programban include-oljuk a headert és letrehozunk egy polargen alakú double-t és ebben történik meg az algoritmus.

6.2. LZW

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

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/binfa.cpp

Magyarázat: A program abból áll, hogy bevesz bináris inputot, majd ezt kiiratja egy fa szerű rendszerben. A fa rendszere úgy működik, hogy egyenként beolvassa a karaktereket, és ha 1-es akkor jobbra hoz létre egy ugyan ilyen gyökér struktúrát. Ez a folyamat addig megy amíg be nem olvas minden karaktert. A kiiratásban be lehet járni az elemeket inorder, preorder, és posztorder sorrendben.

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:https://github.com/BOPI98/BHAX_feladatok/blob/master/preorder_inorder_postorder

Magyarázat: Ebben a forrásban le van írva a három féle bejárási módszer. A preorderben elsőnek kiiratja magát a rootot utána a bal, majd a jobb gyereket. Az inorderben előbb a bal oldalt utána a root, majd a jobb. A postorder bejárási módszer az bal oldallal kezdődik majd jobbal folytatódik és a roottal zárul.

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/BOPI98/BHAX_feladatok/blob/master/lzwt.cpp

A program a harmadik feladat megoldásához hasonló, annyi különbséggel, hogy a binfa kezelőfüggvényeket és eljárásokat a binfa osztályában találjuk meg, majd a node struktúra egy privát részét alkotja az osztálynak. A túlterhelt balra bitshift operátor ellátja a binfa építését, ez hasonlít az insert_tree eljáráshoz.

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/BOPI98/BHAX_feladatok/blob/master/mutatoagyoker.cpp

A program az ezt megelőző programon alapul, a különbség az, hogy a gyökérelemre egy mutatóval mutatunk és egy konstruktorral létre kell hozni a gyökérobjektumot. Ott ahol a gyökérelem referenciáját adta át a függvénynek a program, ott már a gyökérnek az elemét kell átadni a referencia helyett.

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/BOPI98/BHAX_feladatok/blob/master/mozgato.cpp

magyarázat: Ebben a programban a mozgatókonstruktorral úgy épül fel a program, hogy mikor elindul a program akkor lemásilja a kódot és azután fut le a binfa. A mozgatókonstruktor algoritmusa a binfa osztályába kerül be ahol a fa alakját veszi fel. A program többi része ugyan úgy működik, mint az előző feladatokban.

A feladatban segítségemre volt: Nyitrai Dávid

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

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

https://github.com/BOPI98/BHAX_feladatok/blob/master/diagram.png

Tanulságok, tapasztalatok, magyarázat...: A hangyaszimulációs program egy qt-vel létrehozott program ami egy grafikus felületet alkot. Ebben a grafikus felületben fog a program végrehajtódni és a megalkotott algoritmus szerint működni. A program áll egy hangya részből, ami koordinátát tárol és a hangya irányát. Aztán az antthread programmal határozzuk meg a hangyák mozgását és irányát a képernyőn. Az antwin programrészben hozzátesszük a program végét meghatározó eseményeket és a szüneteltetést. A main program össszefoglalja a fő programot majd qt-val létrehozzuk a futtatható programot.

Ebben a programban segített: Nyitrai Dávid

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/BOPI98/BHAX_feladatok/blob/master/eletjatek.java

Tanulságok, tapasztalatok, magyarázat... : Az életjátékot John Conway, egy matematikus találta fel. A játék nem igazi játék de egy érdekes szimuláció. A játékosnak meg kell adnia a kezdő mezőket és a program elkezdi az algoritmus futtatását. Mivel különösebb interakció nincs a programmal így addig fut a program ameddig van élő sejt vagy addig amíg a felhasználó le nem állítja.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

https://github.com/BOPI98/BHAX_feladatok/blob/master/sejtszal.h

https://github.com/BOPI98/BHAX_feladatok/blob/master/sejtszal.cpp

https://github.com/BOPI98/BHAX_feladatok/blob/master/sejtablak.h

https://github.com/BOPI98/BHAX_feladatok/blob/master/sejtablak.cpp

https://github.com/BOPI98/BHAX_feladatok/blob/master/main.cpp

https://github.com/BOPI98/BHAX_feladatok/blob/master/Sejtauto.pro

Tanulságok, tapasztalatok, magyarázat...: A qt programot gui-s magyarán grafikus felületű alkalmazások fejlesztésére használják. Ezt a programot linuxon mac-en windowson és symbian-on lehet használni. A Conway féle életjáték mivel grafikus felületű program ezért a qt-val c++ nyelven is meg lehet írni. Az életjáték négyzetrács mezőiből áll amelyeket körül vesz másik nyolc mező. A folyamat körökre osztott és random vellákat jelölhetünk ki. Ezután a program futása közben ha egy set körül 3 sejt van akkor megmarad. Elpusztul, ha kettőnél kevesebb, vagy háromnál több. És létre jön egy sejt ha egy adott cellának van három szomszédja.

7.4. BrainB Benchmark

Megoldás videó:

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

Tanulságok, tapasztalatok, magyarázat...: A BrainB program egy esport talent mérő program. ez azt jelenti hogy 10 percen keresztül kell a tesztet végezni, amiből megállapítja az alany teljesítményét reflexek szintjén. Egy bizonyos kört kell követni az egérrel ami folyamatosan mozog és mellette még sok más nevű gömb van a zavarás miatt. Minél pontosabban követjük a kört, annál több pontot fog mérni a program. A teszt végén egy szöveges fájlban megtalálhatjuk az eredményt, amely statisztikákat mutat tehetségünkről. Ez a program szintén qt segítségével jött létre a grafikus felület miatt.

8. fejezet

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

Tanulságok, tapasztalatok, magyarázat... : A program arra épül, hogy egy 28x28 képarányú képről felismer egy számot. A képen egy kézzel rajzolt alakból kell, hogy megállapítsa a program ezt a számot. A program elején importáljuk a szükséges függvényeket. Ezután a megfelelő könyvtárból meghívjuk az adatokat és definiáljuk az olvasás és main függvényt. A main függvényben létrehozzuk a "model" változóit, majd a következő részben jön a tesztelés. Az utolsó fázisban lefut a teszt és a kiiratás.

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/melyMNIST.py

Tanulságok, tapasztalatok, magyarázat... : A program elején importálásra kerülnek a szükséges függvények. Ezután négy fő függvényt hozunk létre a programnak. Majd egy ötrétegű neurális hálót hozuk létre a programban. A következő részben a megadott módon kiértékeljük a bemenetet. A tanítási algoritmus megadása után végrehajtjuk lépésenként a training-et. Az utolsó részben a háló tesztelése következik.

8.3. Minecraft-MALMÖ

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

Megoldás forrása: https://github.com/BOPI98/BHAX_feladatok/blob/master/minecraft.py

Tanulságok, tapasztalatok, magyarázat... : A MALMÖ egy olyan projekt melyen keresztül Minecraftban mesterséges inteligenciát lehet létrehozni. A program célja, hogy a karakter akadálymentesen haladjon öt percig, amit az előtte lévő blokkok vizsgálatával próbál elérni. Ezek a cselekedetek ugrásokban vagy irányváltoztatásokban nyilvánulnak meg. A program elején importálni kell a szükséges modulokat beleértve a malmö api fájlt. A missionXML string-ben meg van adva a spawnpoint, pálya generálási szabályok és hogy figyelje a körülötte lévő blokkokat. Az első ciklusban a program vár, amíg készen nem áll az irányításfogadásra, a másodikban pedig elindul a vezérlés. Itt az ugrás szükségét vizsgálja a program. A következőben a karakter váltakozva néz maga körül felderítés céljából. Ezután jön az a tesztelés, ahol eldönti, hogy melyik irányba nézzen. Ebből meg tudja határozni a program hogy mit cselekedjen.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

https://github.com/BOPI98/BHAX_feladatok/blob/master/iterativ.lisp

https://github.com/BOPI98/BHAX_feladatok/blob/master/rekurziv.lisp

iterativ: A program egy faktoriális függvényből áll, ahol sum függvényben mindig összadjuk a bekért x számnak és az x-1 nek a szorzatát. A program végén bekérjük az x-et és lefut a függvény.

rekurzív: A program elején ismét faktoriális függényt hozok létre, ami rekurzív módon számít. A függvény teszteli, hogy az x 1-e ha igen akkor visszatér 1 eredménnyel, de ha nagyobb akkor a sum értéke x*factorial(x-1) lesz. A függvény után 5re lefut a program.

Ebben a feladatban segített: Nyitrai Dávid

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... : Ennek a programnak az a célja, hogy fu scripten keresztül szövegesen tudjuk a gimpnek utasítást adni, amivel egy króm effektet hoz létre. A programban az elsőként definiáljuk a color curve-et 8 tömbben. A következő részben rekurziós függvény van amely egy megadott indexű számot vesz ki a listából. A car adja a lista első elemét a cdr pedig a listát az első elem nélkül. A következőben a szöveg szélességét határozzuk meg, amit felhasználunk a rajzoláskor. Ezek után változókat kell deklarlni, majd a lépéseken keresztül elvégezzük a krómosítást. Létre kell hozni egy réteget, majd bekérünk egy karaktert és a szükséges paramétereket. A lépések lefutása után létrejön egy polinom, aminek tulajdonságait több módon is alkalmazhatunk.

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...: A mandalakészítéshez egy szöveget kell a kép közepére írni, ezt tükrözzük horizontálisan. Ezután ennek másolatát forgatjuk el 90 fokal, majd az eddigiek összevontjának másolatát 45 fokkal. Az eddigiek másolatát pedig végül 30 fokkal és megkapjuk a mandalánkat. A programban az előzőhöz hasonlóan szöveget kell megadni, majd a car és cdr parancsok itt is megjelennek azonos céllal. A függvény deklarálást követően jön maga az algoritmusok lefutása. A felhasználó ilyenkor beírja a szöveget, a betűtípust, a méretet, width-et, height-ot és gradient-et. A megadott paraméterekkel behelyettesítve lefut az algoritmus és megkapjuk a képet.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

2. hét(pici könyv): A programozási nyelvek három szintjét különböztetjük meg, a magasszintű, assembly és gépi nyelvet, melyek közül a magasszintűnek van forráskódja és ehhez szabályok tartoznak, ami a nyelv szintaktikája. A nyelv forráskódját fordítóval elemezni kell szintaktikailag lexikailag, és szematikailag, ezután legeneráljuk a kódot.

3. hét(pici könyv 28. oldaltól): A programozási nyelven belül különböző adattípusok vannak amelyeket névvel azonosítunk, egyes esetekben lehetősége van a felhaszálónak adattípust definiálni. Létezik struktúrált és egyszerű adattípus, ami abban különbözik, hogy az egyszerű nem bontható tovább, mert atomi elemei vannak(a logikában értelmezhetőek literálként). Egy programban minden eszközt deklarálni kell típus név és érték szerint és mindig "beszélő" névvel kell ellátni ezeket a könnyebb későbbi értelmezés érdekében. A változók négy komponensből állnak (név, attribútum, cím és érték). Az azonosító lesz a név, amihez a többi hozzárendelődik. Létezik implicit, explicit és autómatikus deklaráció. Címrendelésnek 3 fajtája van statikus, dinamikus és programozó általi. Értékadásnak két fajtája van, kezdőérték és értékadó utasítás. A C nyelvben egyszerű típusnak számítanak az aritmetikai típusok amelyen alapulnak például a logikai típusok. Az igaz az 1 a hamis a 0.

4. hét(pici könyv) Egy programozási nyelvben jelen vannak a szintaktikai kifejezések, amelyek zárójelekből, operátorokból és operandusokból állnak. Léteznek unáris, bináris és ternárs operátor. Három alakjuk van prefix, infix és postfix. Kifejezésből van típuskényszerítés és típusegyenértékűség.A fordítási időben meghatározandó kifejezés a konstans kifejezés.

5. hét(pici könyv): A programban lévő utasításoknak két csoportjuk van, deklarácós és végrehajtó utasítás. A deklarációs utasítások a fordítóprogramhoz szükséges, a végrehajtó utasításból a kód lesz generálva. A végrehajtó utasításnak van értékadó, üres, ugró, ciklusszervező, elágaztató, hívó, i/o és vezérlésátadó utasításcsoport.

6. hét(pici könyv) A programokban minden eljárás több részre bontható amit program egységeknek nevezünk. A programegységek fordíthatóak közül van olyan amelyeket külön is lehet fordítani, van olyan amelyeket csak eggyüttesen és van amely a kettő kombinációjával lehet fordítani. Az alprogramok olyan programok melyek felhasználhatóak más programokhoz is. Ismétlés egyszerűbb formája egy alprogram meghívása. Egy alprogram névből, formális paraméterlistából, törzsből és környezetből áll. Két féle alprogram van, a függvény és az eljárás. A függvény visszaad egy értéket, az eljárás pedig visszatérő érték nélkül hajt végre műveleteket. Ezekek a műveleteknek szabályai vannak annak érdekében, hogy helyesen

működjenek. Hívásláncalakul ki, ha egy programegység meghív egy másikat és így tovább. Mikor önmagát hívja meg egy függvény azt rekurzív függvénynek nevezzük. Másodlagos belépési pontazt teszi lehetővé, hogy a törzsben is lehessen belépési pontot kialakítani. Paraméterkiértékelés az egy olyan folyamat ahol egy alprogram hívásakor formális es aktuális paraméterek rendelődnek egymáshoz és meghatározódnak azok az információk, melyek a paraméterátadásnál létrehozzák a kommunikációt. A blokk egy programon belüli egység, ami csak a programon belül állhat. A blokknak paraméterei nincsenek, van kezdete törzse és vége.

7. hét(pici könyv): Aktuális-formális paraméterek egymáshoz rendelődése olyan információkat határoznak meg amiket a kommunikációt szolgálják, ezt paraméterkiértékelésnek nevezzük. Név szerinti kötésnél meg kell adni a formális paraméternevet és az aktuális paramétert, itt a sorrend nem számít. Két féle képpen lehet kiértékelni a paramétert, vagy úgy hogy a paraméterszámának meg kell egyeznie vagy úgy hogy az aktuális paraméterszámának kisebbnek kell lennie mint a formálisnak. Paraméterátadásnak van több módja is. Érték szerinti, cím szerinti, eredmény szerinti, erték-eredmény szerinti, név szerinti, szöveg szerinti. Alprogramoknál nem lehet típust átadni paraméterként. Az alprogramnak input, output és input-output paraméterei vannak.

8. hét(pici könyv): 30-33 96-97 98 121 A program blokkja egy olyan programegység amely egy másik programegység belsejében helyezkedik el. Itt is megkülönböztetjük a végrehajtási és deklarációs részt. Egy programban a név hatásköre alatt értjük azt a programrészt ahol a program jellemzői azonosak. A hatáskörkezelés jelenti azt a tevékenységet amikor megállapítjuk egy név hatáskörét. A fordítás közben megtörténő hatáskörkezelés alapja a programszövegnek a programegység szerkezete. Ha talál egy szabad nevet egy programegységben akkor kilép és ellenőrzi, hogy lokális-e a név. Ha az akkor vége, de ha nem akkor tovább lépked ki addig amíg a lokális nevet meg nem találja. A legkülső szinten van két eset. Az egyik esetben hibát dob a deklaráció hibájára, a másik pedig autómatikus deklarációt végez és hiba nélkül fut le a program. A hatáskör csak befelé terjed. Azt a nevet nevezzük globálisnak amely az adott programegységben nem lokális, de látható onnan. A dinamikus hatáskörkezelést a futtató rendszer végzi és a hívási lánc az alapja. Az eljárásorientált nyelvek megvalósítják a hatáskörkezelést.

9. hét(pici könyv 134-138)Az I/O a programnyelvben az az eszközrendszer, ami kommunikációt létesít a felhasználó és a program között. Ez azt jelenti, hogy a perifériákról beviszi az adatot a programba miközben fut. Az I/O-nak van egy központja melyet állománynak nevezünk. Logikai állománynak a névvel rendelkező programozási eszközöket nevezzük ezek rendelkeznek attribútumokkal az állományjellemzés céljából. Fizikai állomány célja, hogy a perifériákon megjelenő adatot tartalmazza. Egy állomány lehet input, output, input-output állomány. Az input állomány már a feldolgozás előtt létre kell, hogy jöjjön, majd ezt beolvasva fut le a feldolgozás. Az output a feldolgozás után visszaadott érték ahova csak kiírni lehet. Az input-output egyszerre van jelen a feldolgozás előtt és után, szóval lehet írni és olvasni is, közben változhat a tartalma. A periféria és a tár közötti adatátvitel történhet két módon. Az egyik konvertálással a másik konvertálás nélkül történik. A folyamatos módú bevitelnél a perifériáról beolvasott adatsorozatot típushoz mérten megfelelően kell dekódolni. A nyevleknél van erre három eszközrendszer, ami a formátumos módú, szerkesztett módú és listázott módú adatátvitelből áll. A programon belüli állományokkal való munkához feltételekhez kell igazodnunk. Az első a deklaráció amivel megnevezzük milyen állományokkal dolgozunk. A második az összerendezés, melynek során egy fizikai állományt feleltetünk meg a logikai állománynak. Harmadikként jön az állomány megnyitás, ahol az operációs rendszer rutin ellenőrzése fut le. Az ezt követő feldolgozás a megnyitott állomány olvasását, vagy abba írását hajtja végre. A lezárásnál ugyanúgy rutinokat aktivizál az operációsrendszer, majd lezárással megszűnik a kapcsolat. Implicit állománynak azt nevezzük, mikor a programozó közvetlenül a perifériával ír, olvas. Fortranban szeriális, szekvenciákis, ditekt állományt tud kezelni, fix rekordformátumot tud kezelni. Cobolnál mindíg van konverzió, szeriális, szekvenciális, direkt, indexelt, invertált állományszerkezetet ismert. Fix rekordformátumot kezel és lehetséges a blokkolás.

PL/L-ben az összes állományszerkezetet, adatátviteli módot, rekordformátumot, blokkolási lehetőséget kezeli. Pascalban blokkolás nincs, nincs i/o, csak beépített alprogram van. C-ben az i/o nem a nyelv része, standard könyvtári függvények vannak, bináris átvitel és folyamatos átvitel van, az i/o függvények teszik lehetővé az írás, olvasást. Az ada minden perifériát tud kezelni, i/o-t csomagokkal valósítja meg.

11.hét(pici könyv): 112 113 A kivétel kezelés egy olyan funkció, amellyel kivételek bekövetkezése esetén megszakítást okoz a program. Ezzel az operációs rendszer helyett program vezérelt lesz a megszakítás. Egyes események figyelése letiltható, de ez annyit jelent, hogy a programi szinten ignorálja a kivételkezelést, ami problémát okozhat a program működésében. A kivételkezelés megtalálható a PL/I és Ada nyelvben. A PL/I nyelvben a kivételkezelés úgy működik, hogy amikor a program futása közben bekövetkezik egy kivétel, a program megkeresi a kiváltó okot, megszünteti a szituációt és visszatér a program normál működéséhez. Az Ada ezzel ellentétben egy speciális szituáció bekövetkeztekor abba hagyja az adott tevékenységet és olyannal folytatja ami elkerüli a kivétel kiváltódás helyét.

1. Milyen beépített kivételek vannak a nyelvben? :Lehet alapvető matematikai ellentmondás, memóriaelféréssel kapcsolatos hiba.

2. Definiálhat-e programozó saját kivételt? :Igen.Minden programhoz lehet saját kivételeket definiákni.

3. Milyenek a kivételkezelő hatásköri szabályai? Abban a hatáskörben működik a kivételkezelő, amelyikben az meg lett hívva.

4. Hogyan folytatódik a program a kivételkezelés után? Az attól függ, hogy milyen hibát talál. Van amitől le kell, hogy álljon, viszont van, ahol kezelés után futhat tovább.

5. Van-e a nyevlben beépített kivételkezelő? Igen, az első válasz példáival.

11.hét(pici könyv 2): 38 Amikor megváltoztatjuk egy példánymódszer implementációját, magyarán újraimplementáljuk, annak kompatibilisnek kell lenni a felette elhelyezkedő örökölt módszerrel. Ez azt takarja, hogy specifikációjukban meg kell egyezniük, csak azokat a kivételeket válthatja ki melyet az örökölt módszer és csak enyhíteni lehet a bezárást. Javaban egy meghívás esetén a névhez tartozó kód fog meghívódni. Javaban minden tag öröklődik, és ezzel együtt nem lehetséges, hogy öröklődés során elmaradjon egy tag. Öröklődés során újra lehet implementálni a módszereket és át lehet definiálni az adattagokat, viszont nem szüntetheti meg azt az örököltetett. Példányosítás során alapállapotot kell létrehozni, amelyek lehetségesek paraméterekkel, de erre a célra szolgálnak a konstruktorok, melyek típus nélküli módszerek. A konstruktorok példányosításánál a meghívás és inicializálás autómatikus. A konstruktorok jellemzője még, hogy new operátor mellet lehet őket meghívni és nem örökölhetők.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

5. hét(k and r könyv): Minden C programban egy utasítást pontosvesszővel zárunk le, kapcsos zárójellel összefogjuk őket. Egy logikai kifejezésnél igaz érték esetén az azt követő kapcsos zárójelben lévő utasítás fut le. Hamis érték esetén else után írt utasítások futnak le. A for és while ciklusok olyan függvények, amelyek egy logikai tesztelés után hajtanak végre utasításokat addig amíg a feltétel igaz. Vannak elöl és hátultesztelő ciklusok(pl: do-while). A ciklusban használhatunk kilépő parancsot is ami a break, viszont ha több ciklusból akarunk kilépni, akkor használhatunk goto utasítást. Minden utasítás leírási sorrendben fut le, viszont egy összetett utasítást is egy utasításként fog kezelni.

10.3. Programozás

[BMECPP]

5. hét(BME C++): A C++ nyelv a továbbfejlesztett változata a C-nek. A C-ben például nincs string változó csak karaktertömb. Referenciatípus viszont egyáltalán nem deklarálható a c nyelvben csak a C++-ban. A c-ben egy függvényt neve alapján azonosítunk, ezzel ellentétben c++-ban az argumentumokkal együtt azonosítja a függvényt. C++-ban létezik változó szerinti paraméterátadás ami lehetővé teszi a műveletek elvégzését úgy hogy a változó értékét ez nem befolyásolja.

6. hét(BME C++) 15-59 Az objektum orientáltság alapelvei rávilágítanak egy tisztább, rendezettebb és logikusabb struktúrára. Az objektum orientált adatstruktúrát osztályoknak, vagy class-oknak nevezik, ami egyfajta kategóriát definiál. Egy ilyen osztálynak lehetnek elemei, melyeket objektumnak nevezünk. Ezek szintén rendelkeznek önmeghatározó tulajdonságokkal. Egységbe zárással létre hozhatunk például koordináta rendszert szimuláló programot, ahol egy külön struktúra lehet egy pont x és y értékekkel. Ezeket a pontokat műveleti függvények segítségével is meghatározhatjuk. A struktúra adattagjai, másnéven tagváltozók a struktórára vonatkozó változókat jelentik, melyek azt jellemzik. Az adatrejtés lehetővé teszi, hogy egy struktúrán belül priváttá azaz csak azon belül láthatóvá tegyük vagy pedig publikká tegyük, ami engedi, hogy a tagváltozók és függvények más struktúrából is hozzáférhetőek legyenek. Objektumorientált programnál fontos lehet, hogy a tagváltozók szabályszerűen legyenek megadva, ezeket a tagfüggvényekkel tesztelve meg lehet oldani, de konstruktorok segítségével a tagfüggvények a létrejöttükkor saját magukat inicializálják. A destruktor az objektum esetleges megszűnésekor jön szóba, amikor erőforrás felszabadítás szükséges. Dinamikus memóriakezelésért felelősek a malloc és a free függvények, de c++-ban azonban már operátorok végzik a memóriafoglalást, mint például a new és a delete. A másolókonstruktor célja a másolás, amit az előzőkhöz hasonlóan a már megévő objektumok alapján inicializálva hoz létre. A friend függvény jogot ad globális függvényeknek egy osztályon belüli védett tagfüggvények hozzáféréséhez. Ennek létezik egy osztály verziója ami ugyan erre a célra szolgál.

7. hét(BME C++) C-ben az operátorok feladata, hogy argumentumok műveleteit végezze el, az eredményt visszatérési értékkel kapjuk meg. Operátorok közötti prioritás szabályrendszer szerint dől el. Ha egyenrangú operátotokkal dolgozunk ajánlott záróójelek használata bizonyos helyzetekben. Az operátor és a függvény nem azonos mivel a függvény nem képes mellékhatásokra az érték szerinti paraméterátadás miatt.

8. hét(BME C++) A sablon egy olyan osztálysablon, vagy függvénysablon amelyet paraméterként használunk. Ennek az az értelme, hogy akármilyen típusú elemet lehet tárolni a sablonban. Ha például egy rendezést akarunk végrehajtani több féle típussal, akkor elég a sablon paramétereket használva tudunk akár karaktereket, egész számokat, vagy double-t is rendezni egy függvénnyel. A sablonparaméter típuson kívül lehet még típusos konstans, melyet például square függvényhez lehet hasznosítani. A square csak konstans paraméterrel működik, mivel a sablonnak a kifejtése fordítás közben történik. A square függvény használata kevesebb forítási időt igényel. Az intfifo osztályt osztálysablonná úgy lehet átalakítani, hogy template classként definiáljuk az osztályt, ezzel jelezzük a fordítónak, hogy ez nem közönséges osztály. Ezután sablonparamétert kell bevezetni. Az osztálynevek helyett osztálynév és sablonparaméterek szerepeltetését kell megvalósítani.

11.hét(BME C++): A hagyományos hibakezelés alapjait részletezve hívási láncok szemléltetését mutatja be a fejezet első részében a könyv. Egy programban a main függvényben hibakezelést láthatunk, ebből arra következtetünk, hogy hátrányos hibakódokra épülő megoldást használni. A következő részben a kivételkezelés alapjairól tudhatunk meg többet. A programrész szemléltet egy olyan eljárási módot ahol, ha a program talál egy hibát, azonnal a hibakezelő részre ugorjon. Ha kivételezve van akkor a kivételkezelőre

ugorjon. Ezután pontokba összeszedve láthatjuk a kivételkezelés mechanizmusainak alapjait. A következő részben egymásba ágazott try-catch blokkokról olvashatunk, amiből megtudhatjuk, hogy így több szinten tudjuk kezelni a kivételeket. Az ezt követő részben az újradobásról van szó, amelyet kódcsipettel szemléltet.

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;
}
```

A bekért számot a d-ben tároljuk el, majd ezt ellenőrizzük, hogy nem-e nulla. Ha a d változó értéke egyenő nuléűval akkor a throw-val hibát dob ki a program. Viszont ha ez nem teljesül, megy tovább a program, meghatározzuk a reciprokot és kiiratjuk. A throw általi hibát catchel el lehet kapni és az if-ben megadott mondatot kiírjuk.

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";
    }
```

A try-catch blockot egymásba lehet ágyazni, melyenek lényege, hogy egy adott kivételeket alacsonyabb szinten kezeljünk. Az elkapott kivételt újra lehet dobni throw-val. Visszacsévelés az amikor egy kivétel dobásánál annak elkapásáig a függvények hívási láncában a függvényeknek a lokális változói felszabadulnak felfelé haladva.

```
Erőforrás kezelés:
    class MessageHandler
    {
     public:
        void ProcessMessage(istream& is)
        {
          Message *pMessage;
          //Következő üzenet beolvasása.
          while((pMessage = readNextMessage(is)) != NULL)
          {
            try
            {
              //Kivételt dobhat!
              pMessage->Process();
              // ...
              // Ha végeztünk, felszabadítjuk a Message objektumot.
              delete pMessage;
            }
            catch(...)
            {
              delete pMessage;
              throw;
            }
          }
        }
     private:
       Message* readNextMessage(istream& is)
       {   ... }
    };
```

A program egy bemeneti fájlból olvas be üzeneteket egy osztály segítségével. Ebben az osztályban egy tagfüggvény addig olvassa be az objektumokat ameddig null-ra nem tér vissza. Miután az olvasó függvény beolvassa az üzenetet, majd az objektum feldolgozta az üzenetet, az üzenet objektumja felszabadul. Ha ilyenkor a feldolgozó objektum kivételbe ütközik, akkor nem szabadul fel az üzenet objektum. Ezt a problémát tudjuk orvosolni try-catch-el, elkapjuk a kivételt és felszabadítjuk a lefoglalt memóriát, majd újradobjuk a kivételt.

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

0. heť - „Hello, Berners-Lee!”

11.1. C++ és Java összehasonlítása (Szoftverfejlesztés C++ nyelven, Java 2 útikalauz programozóknak)

A Java és a C++ két különböző fordítási metódussal fut le. A Java egy interpreterrel hoz létre a Java Virtual Machine számára kezelhető kódot. A C++ pedig egy forrás kódból fordít le egy futtatható gépi kódot. A java előnye ezzel szemben az, hogy a Java Virtual Machine bármiylen platformon le tudja futtatni ugyan azt a kódot, nem úgy mint a C++. Egy C++ és egy Java program felépítése meglehetősen különbözik. Egy Java kód main függvénye egy main class-ban van rendezve, hiszen a metódusoknak és változóknak egy osztályhoz kell tartoznia, egy C++ programban pedig egy main fájl tartalmazza a main függvényt. Megfigyelhetjük, hogy a Java main függvénye egy void, melynek paraméterei string-ben vannak megadva, ezzel ellentétben a C++ kód egy int visszatérő értékű main függvénnyel rendelkezik és karaktertömbben vannak a paraméterek tárolva.

Java-ban vannak primitív és nem primitív típusok. A kettő között az a különbség, hogy a primitív az értéket tárolja, a nem primitív pedig egy objektumra hivatkoznak. Egy primitív int-et például lehet Integer csomagoló osztállyal helyettesíteni, amely autómatikusan csomagol be és ki az 5-ös Java verzió óta. Javaban nincs karakter változó, ezért egy beépített String osztályt használunk. A Java-ban a tömbök nem pointerként vannak jelen, mint a C++-ban hanem külön típusként. Mivel típusként vannak jelen, így tartozik hozzá méretlekérdező függvények, így nincs szükség változókra a tárolásához. Java-ban az osztályok class szóval tudjuk deklarálni, melyek az Object osztály leszármazottja, viszont egy osztályban nem lehetséges többszörös öröklődés. Szülő osztályra referenciával lehet hivatkozni. Egy C++ programban használatos egy destruktor, viszont Java-ban erre nincs szükség. A másoló konstruktorok helyettesíthetőek clone metódussal. Azok az osztályok támogatják a másolást, melyek a Cloneable interfészt implementálják. Javaban példányosításnál objektumhivatkozást hozunk létre, ami egy objektumra mutat. A Java-ban fontos tény, hogy referenciák vannak, nincsenek pointerek. Újdonságként jelennek meg a Java-ban a generikusok, melyek a sablon osztályt helyettesítik. Egy osztály örökölhet tagfüggvényeket a szülőosztályától, egyes esetekben ezeket felül is kell definiálni. Ezt a C++-ban virtual-lal tudtuk megoldani. A többitől eltér abban, hogy nem szükséges az objektum típusát explicit módon megadni.

11.2. Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven

A Pyton nyelvet Guido van Rossum alkotta meg 1990-ben. A pyton nyelv alapvetően egy könnyen tanulható programozái nyelv , azonban elég komoly szinteket is el lehet vele érni kezdő szinten is. A nyelv a C++ vagy java-val szemben nincs szükség fordításra. A forráskódból azonnal képes lefutni, mivel scriptnyelv. Prototípusok tesztelésére kiváló lehetőséget nyújt mellyel c,c++ és java programok funkciók és eljárások tesztelhetőek. A standard pyton kódkönyvtár segítséget nyújt fájlkezelésben , hálózatkezelésben és rendszerhívásokban. Egy pyton programban érdekesség, hogy sor végén nem kell ";"-ot rakni, szintaxisa egyszerű , behúzások alapján választja el a blokkokat. Így kapcsoszárójelekre nincs szükség. Egy hosszú utasításnál, ha nem fér bele a kód egy sorba, akkor "\"-el ezt lehet jelölni. A megjegyzések # karakterrel működnek, ami a sor végéig hat. Tokeneknek négy különböző fajtája van, azonosító, kulcsszó, delmiter és literál A C++,C, java nyelvekhez hasonlóan az adatokat objektumok reprezentálják és az azokon végrehajtható műveletek az objektum típusától függenek. A pyronban adattípusok lehetnek számok logikai értékek, halmazok, listák, stringek, szótárak. A változók objektumra mutató referenciák. A pytonban ha egy objektumot és hivatkozását törlünk, egy garbage collector felszabadítja az annak lefoglalt helyet. Ugyanúgy mint más nyelvben, a pytonban is megtalálhatóak a megszokott cilusok, if elágazások. A nyelvben szintén elérhető a goto funkció, amely az adott label-ekhez ugrik. Kivételkezelést is tartalmaz a nyelv, mint például a try, except vagy raise. A pyton fejlesztéseket megkönnyítő modulokat tartalmaz, ezek között rengeteg olyan van melyek a mobil fejlesztést segítik. Felhasználói felület kezelését biztosítja, hálózatkezelést segíti és kamera is elérhető. A sysinfo modullal a telefon adatait lekérdezhetjük, audio modul segítségével pedig hangfelvételeket hozunk létre. Ezek mellett saját modult is létrehozhatunk melyet a programunkba importálhatunk.

12. fejezet

1. heť - „Hello, Arroway!”

12.1. OO szemleĺet

A moďosiťott polařtranszformaćioś normaĺis generařor beprogramozaśa Java nyelven. Mutassunk ra, hogy a mi termeśzetes sajať megoldaśunk (az algoritmus egyszerre keť normaĺist aĺliť elo, kell egy peĺdańytag, amely a nem visszaadottat tařolja eś egy logikai tag, hogy van-e tařolt vagy futtatni kell az algot.) eś az OpenJDK, Oracle JDK-ban a Sun aĺtal adott OO szerveześ ua.! Ugyanezt iŕjuk meg C++ nyelven is! (laśd meǵ UDPROG repoː source/labor/polargen)

Megoldás forrása: linkhttps://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf (16-22 foĺia)

linkhttps://github.com/BOPI98/prog2_feladatok/blob/master/Main.java

link

Tanulságok, tapasztalatok, magyarázat... : A program azt csinálja, hogy generál két számot, és ezek közöl az egyiket eltárolja, a másikat pedig visszaadja. Mivel elsőnek azt vizsgálja, hogy van e már tárolt elem, ezért kell egy "nincsTárolt" boolean(igaz-hamis értékeket tároló változó). Ha van benne elem akkor egy double-ban eltárolja. Ennek csak annyi az értelme, hogy nem generál új számot, ha már van egy tárolt. A program objektum orientáltan van megírva, ugyanis maga a polargen osztály egy külön "fájlban" van megírva és a main program ezt használja a lefutáshoz.

main.java:

```
package polargen;

import polargen.PolarGen;

public class Main {
    public static void main (String args[]) {
        PolarGen polarGen = new PolarGen();
        for (int i = 0; i < 10; i++) {
            try {
                System.out.println(polarGen.Next());
            } catch (Exception e) {
                System.out.println("There is no number");
```

```
            }
        }
    }
}
```

polargen.java:

```
„package polargen;

import java.util.Random;

public class PolarGen {

    //Variables
    private boolean hasStoraged = false;
    private double storagedNumber;



    public double Next() throws NullPointerException {
        if(!hasStoraged){
            double u1,u2,v1,v2,w;
            Random random = new Random();
            do {
                u1 = random.nextDouble();
                u2 = random.nextDouble();
                v1 = 2 * u1 - 1;
                v2 = 2 * u2 - 1;
                w = v1 * v1 + v2 * v2;
            } while (w > 1);

            double r = Math.sqrt((-2 * Math.log(w))/w );

            storagedNumber = r * v2;
            hasStoraged = !hasStoraged;

            return r * v1;
        } else {
            hasStoraged = !hasStoraged;
            return storagedNumber;
        }
    }
}”
```

12.2. Homokozó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutassunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden máris működik (erre utal

a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen). Miután már áttettük Java nyelvre, tegyük be egy Java Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját!

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/prog2_feladatok/tree/master/scr/servlet

Tanulságok, tapasztalatok, magyarázat... : A binfa a könyv az első részének egy feladata amelyről a részletek megtalálhatóak. C++ nyelvről az átírása abban nyílvánul meg, hogy a pointereket, referenciákat a java kódból mellőzni kell. Ezek mellett a java kód struktúrájának megfelelően kell a programot átalakítani.

A feladat arról szól, hogy a binfát le lehessen kérni böngészőből a GET kéréssel. Ez a Java Servlettel működik eclipse-el és Tomcat szerverrel. GET-tel létrejön a binfa objektum, az input szövegből elkészül a bemeneti bináris kód és felépül belőle a fa, ezután a bejárási mód szerint bejárja a fát. Ha a paramétereket nem adtuk meg az elején a program default adatokkal fogja azokat kitölteni.

12.3. "Gagyi"

Az ismert formális2 „while (x <= t && x >= t && t != x);” tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására3 , hogy a 128-nál inkluzív objektum példányokat poolozza!

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/prog2_feladatok/blob/master/Gagyi.java

https://github.com/BOPI98/prog2_feladatok/blob/master/Gagyi_vegtelen.java

Tanulságok, tapasztalatok, magyarázat... : A felaladat alapvetően egy argumentumra épül, ami egy while feltételében ciklusban található. A program lényege, hogy ugyanazzal a ciklussal tudjunk egy végtelen ciklusú programot és egy rendesen lefutót megírni. Ezt az int változó pool-jával értem el ugyanis ha a poolon kívül esik a véltozó értéke akkor a ciklus már végtelen lesz. Azonban ha bele esik akkor normálisan fut a program. A -128 nál kisebb értékeket már nem tartalmazza a java integer poolja és ezért ha mind a két változónak ugyan azt az értéket adjuk és lefuttatjuk a programot, végtelen ciklust fogunk kapni.

12.4. Yoda

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

Megoldás videó:

Megoldás forrása: https://github.com/BOPI98/prog2_feladatok/blob/master/Yoda.java

Tanulságok, tapasztalatok, magyarázat... : A yoda stílusban íródott kódban az a lényeg, hogy egy kifejezésben meg vannak cserélve a tagok. Általába ez nem a megszokott módon szokott kinézni ezért ilyen érdekes vagy különleges. Az elnevezése a Star Wars univerzumból ismert karakterről kapta a nevét, azért mert ő is hasonlóképpen alkotta a mondatait. A mondatokat mindig tárggyal kezdte és utána az alany állítmányt.

Ez a beszédstílusa azonban inkább az angol verzióban érezhető, mivel a magyar nyelvű filmekben más a szinkron. Ennek a stílusnak nem igazán van sok előnye viszont kritikusok szerint elég rosszul olvasható és értelmezhető egy ilyen kód.

12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbpalg.pdf és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadsz, de csak végső esetben: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitokjavat/apbs02.html#pi_jegyei (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... : A BBP vagyis a Bailey-Borwein-Plouffe egy 1995-ben kitalált algoritmus, ami a Pi hexadecimális jegyeinek egy d>0 számjegytől való kiszámolására szolgál. Ez az algoritmus: {16dπ} = {4{16dS1} - 2{16dS4} - {16dS5} - {16dS6}} A kód: https://github.com/BOPI98/-prog2_feladatok/blob/master/bbp.java

13. fejezet

2. heť - „Hello, Liskov!”

13.1. Liskov helyettesítés sértése

Írjunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés.

Megoldás videó:

Megoldás forrása: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/BatfaiBarki/-madarak/)

linkhttps://github.com/BOPI98/prog2_feladatok/blob/master/madar.java

Tanulságok, tapasztalatok, magyarázat... : Barabara Liskov egy amarikai informatikus volt aki kidolgozta a liskov helyettesítési elvet. Ez az elv arról szól, hogy egy T objektum S altípusát minden olyan helyre behelyettesíthetjük, ahol T szerepel, mindezt a programrész tulajdonságainak megváltoztatásai nélkül. Ezt úgy kell elképzelni, mint például a madarakat. Ha létre hozunk egy madár típust és a madár tud repülni, akkor az annak egy altípusa, egy sas is tud repülni. Itt következik az elv megsértése, mikor ugyanis egy madarak csoportjába tartozó pingvinre cseréljük ki az altípust. Ezúttal viszont az a probléma, hogy a pingvin nem tud repülni. Ezt a problémár orvosolni tudjuk azzal hogy a madarakat két osztályra bontjuk, repülésre képes és képtelen madarak típusára.

13.2. Szülő-gyerek

Írjunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek!

Megoldás videó:

Megoldás forrása: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)

linkhttps://github.com/BOPI98/prog2_feladatok/blob/master/szulogyerek.java

Tanulságok, tapasztalatok, magyarázat... : Egy programban a szülő osztály metódusai örököltek a gyermek osztályban, azonban fordítva ez nem lehetséges. Ez a polimorphizmus szabálya miatt van. Ennek a feladatnak a lényege, hogy ezt demonstráljuk. A program első lépése az, hogy létrehozunk egy szülőnek szánt

osztályt egy metódussal. Aztán egy gyerek osztályt amely a szülőből származtatik. Ezt az "extends"-el tudjuk javaban létrehozni. A main függvényben példányosítjuk a szülőn keresztül a gyerek osztályt. Ha például a szülőn keresztül szeretnénk a szülő metódusait akkor természetesen működni fog, viszont ha a gyerek metódusait szeretnénk használni a szülőn keresztül az már nem fog működni. Szintén nem működik abban az esetben ha két gyerek egymás metódusait akarját használni, viszont a fán felfele haladva (gyereken keresztül szülő metódusait) lehet használni.

13.3. Anti OO

A BBP algoritmussal4 a Pi hexadecimális kifejtésének a 0. pozíciótól számított 106, 107, 108 darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! https://www.tankonyvtar.hu/hu/tarta tanitok-javat/apas03.html#id561066

Megoldás videó:

Megoldás forrása: linkhttps://github.com/BOPI98/prog2_feladatok/blob/master/bbp.c

Tanulságok, tapasztalatok, magyarázat... : Az előző fejezetben a bbp kód már át van beszélve java-ban. Ebben a feladatban c, c++ és c# nyelven is le kell tesztelnünk. A számításokat a java nyerte nagy fölénnyel, az eredmény 224.652 volt, amíg a c++ már 274.561 eredményt adott.

0xJegy	**C**	**C++**	**C#**	**Java**
10^6	1.879098	1.901872	2,064295	1.69
10^7	20.235581	24.445074	22,2933308	20.235123
10^8	243.842315	223.834862	234,5261652	234.845344

13.1. táblázat. Eredmények

13.4. deprecated - Hello, Android!(zöld)

Élesszük fel a https://github.com/nbatfai/SamuEntropy/tree/master/cs projektjeit és vessünk össze néhány egymásra következőt, hogy hogyan változtak a források

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.5. Hello, Android!

Élesszük fel az SMNIST for Humans projektet! https://gitlab.com/nbatfai/smnist/tree/master/forHumans/SMNIST Apró módosításokat eszközölj benne, pl. színvilág.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.6. Hello, SMNIST for Humans!(piros)

Fejleszd tovább az SMNIST for Humans projektet SMNIST for Anyone emberre szánt appá! Lásd az smnist2_kutatasi_jegyzokonyv.pdf-ben a részletesebb hátteret!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

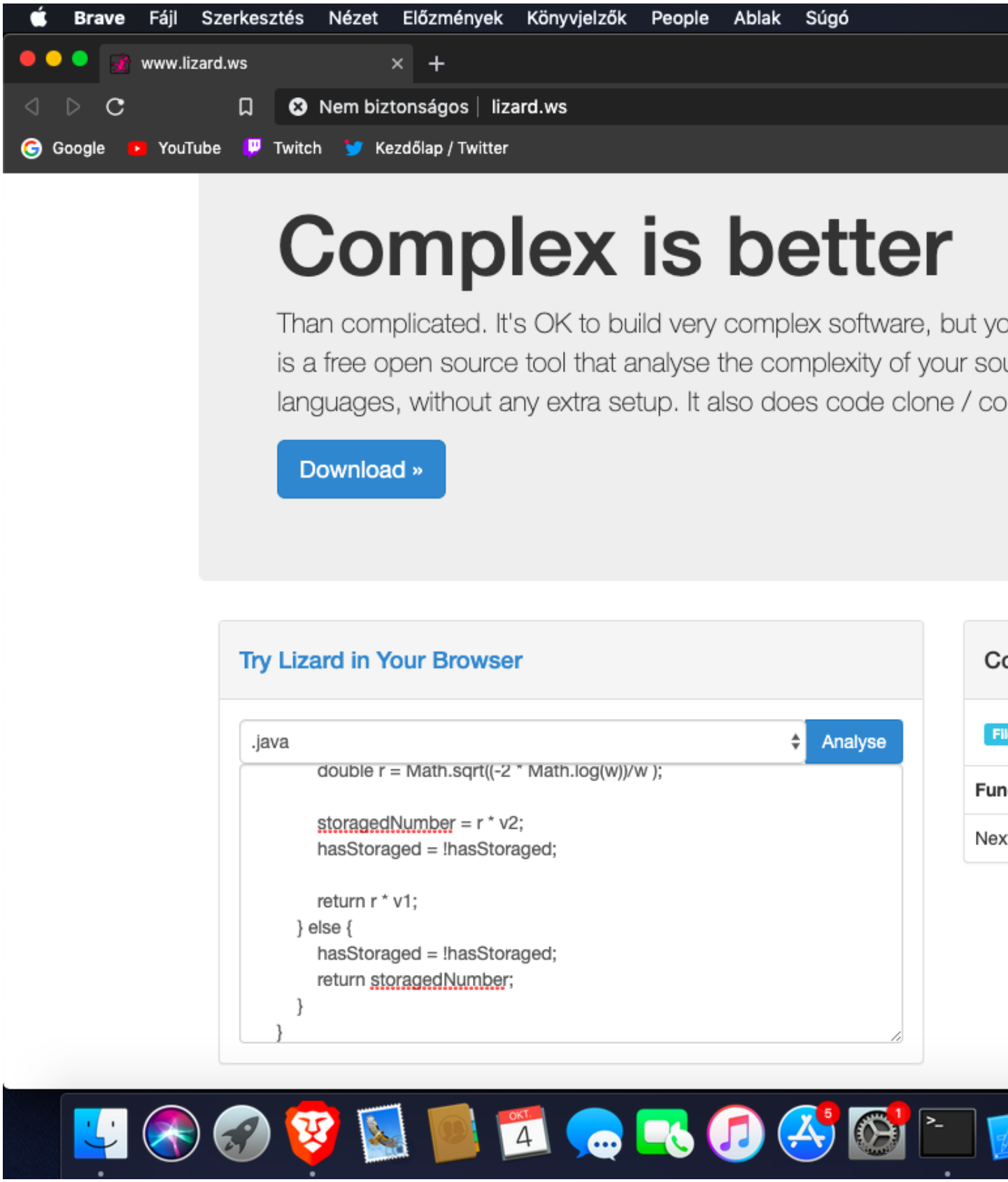
13.7. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalom tekintetében a https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf (77-79 fóliát)!

Oldal: link

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... : A ciklomatikus komplexitás vezérlési gráfban a független utaknak a maximális száma. Két út akkor függvetlen ha létezik mindkettőben olyan pont vagy él, amely a másik útnak eleme. Az értéke: M=E-N+2P, ahol E az élek száma, N a csúcsok száma, P pedig a pontok száma. Ennek a komplexitásnak a kitalálója Thomas McCabe.



A polártranszformáció számítási részét beillesztettem java nyelven, majd eredményként kiírta a komplexitást 3-ra, az NLOC 20, és a tokenhez 134-re.

14. fejezet

3. heť - „Hello, Mandelbrot!”

14.1. Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nlERlEOs. https://arato.inf.unideb.hu/batfai.norbert/UD (28-32 fólia)

Megoldás forrása: link https://github.com/BOPI98/prog2_feladatok/tree/master/Mandelbrot

Tanulságok, tapasztalatok, magyarázat... : A feladatnak az a lényege, hogy egy c++ kódból automatikusan generáljunk egy uml diagramot. Erre a feladatra használhatunk eclipse-et is egy plugin segítségével, vagy bármely más programot. Én az Umbrello programot használtam, ami egy open-source program és tökéletesen megfelel a célnak. A program telepítése után importálhatjuk a c++ programunkat az import wizard segítségével. Az importálás után, már csak annyi dolgunk van, hogy az osztályokat behúzzuk a rajzterületre és automatikusan elkészül a diagram, kapcsolatokkal együtt.

14.2. Forward engineering UML osztálydiagram

UML-ben tervezzünk osztályokat és generáljunk belőle forrást!

Megoldás forrása: link https://github.com/BOPI98/prog2_feladatok/tree/master/Mandelbrot/Forward

Tanulságok, tapasztalatok, magyarázat... : A feladat arrol szól, hogy egy elkészített uml diagramot kell kóddá alakítani. Ezt szintén meg lehet csinálni umbrello-val, de más program is alkalmas rá. Umbrelloban a code fül alatt található code generation wizard segítségével a meglévő objektumokat kóddá lehet konvertálni a megadott helyre.

14.3. Egy esettan

BME C++ könyv 14. fejezet (427-444,445-469)

Ebben a fejezetben az UML és a C++ kapcsolata lesz tárgyalva. Az UML egy általános nyev, amely jelenleg minden fejlesztési nyelv szabványa, ezáltal minden eszközt támogat. Az UML nem csak objektumorientált

programokról tud modelt készíteni. Az UML egy grafikus modellezőnyelv, melyben a diagramok dobozokból, vonalakból, ikonokból és szövegekből állnak. Ennek a grafikonnak az a lényege, hogy egy tömör vázat kapjunk az adott programról. Ez a grafikus megjelenítés, akkor hasznos, mikor egy problémát magas szinten akarunk megoldani és nem szeretnénk a részleteket figyelembe venni. Az osztálydiagramm legfontosabb alkotóelemei az osztályok, melyeket téglalap jelöl. Ennek 3 fő szakasza van: az osztálynév a legfelső, az attribútumok a középső, a műveletek pedig a legalsó szakaszban vannak. Az attribútumok a tagváltozóknak, a műveletek pedig a függvényeknek felelnek meg. Az osztály attribútumait ezzel a szintaxissal adjuk meg: láthatóság név: típus multiplicitás = alapértelmezett érték{tulajdonság}. A láthatóságot négy féleképpen lehet megadni, lehet public, private, protected és package, viszont az utóbbi a c++ nyelvben nem lehetséges. A név és a láthatóság között elhelyezett / jel származtatott attribútumot jelöl. Ha egy programban más változókból származtatik egy változó értéke, akkor azt a c++ kódban nem jelöljük. Ez akkor fontos ha egy osztály egy objektumát relációs adatbázisban szeretnénk tárolni, ahol a lekérdezés során számoljuk ki az értéket. Ha tömb az attribútum akkor a tartalmazott elemek lehetséges számának a halmazát a multiplicitás adja meg. A multiplicitás egy szám vagy egy egészekből álló intrvallum, melyek szövegként megjeleníthetőek. Az alapértelmezett értéke egy. Paramétereknél ha nem írunk értéket akkor az in az alapértelmezett. Az in az érték szerinti paraméterátadásnak felel meg, az inout jelentése, hogy a függvény felhasználja a referencia vagy pointer által jelzett objektumot, majd megváltoztatja azt. Az out esetében csak megváltoztatja az objektumot. A statikus műveletek aláhúzással vannak jelölve, a tisztán virtuális függvényeket dőlt betűvel vagy {abstract} tulajdonsággal adjuk meg.

Az egyfajta kapcsolatot üres háromszögfejű nyíllal jelöljük. A szerepnév előtt - jel van, ami a szerepnévből képződő tagváltozó láthatóságát jelzi. A sablonok ki kifejezését vagy a példányosítását kétféleképpen lehet lemodellezni. Az egyszerűbb jelölés az osztály neve után kettősponttal elválasztva feltünteti a példányosítást. Ezek a modellező eszközök képesek modellből előállítani kódot és fordítva. Amikor modellből kódot állítunk elő, azt kódgenerálásnak hívjuk, és amikor kódból modellt azt pedig kódvisszafejtésnek nevezzük. Ezt a két szolgáltatást együtt képirányú generálásnak nevezzük.

14.4. BPMN

Rajzoljunk le egy tevékenységet BPMN-ben! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog
(34-47 fólia)

Megoldás forrása: https://cawemo.com/

link kép

Tanulságok, tapasztalatok, magyarázat... : A feladat az volt, hogy rajzoljunk BPMN diagrammot egy program segítségével. Én a diagrammot a cawemo.com-on csináltam , mivel így nem szükséges letölteni külön programot és felhőből is elérhetőek a projektek. A használata egyszerű, felhasználóbarát és gyorsan lehet vele dolgozni. Az én diagrammom egy egyszerű fej vagy írás programot demonstrál egy szerver és egy felhasználó között. A szerver az induláskor bekér egy értéket(fej vagy írás), ezután generál egy random számot nulla és egy között , majd egy tesztelés által eldől, hogy a játékos nyert vagy veszített. Az eredmény megszületése után az eredményt elküldi a kliensnek, majd újra inputra vár. A kliens ennél egyszerűbb ugyanis annyit csinál, hogy bekér egy értéket amit elküld a szervernek, majd várja az eredményt. Ha megkapta az eredményt előről kezdődik a program.

Ezzel a programmal azonban nem csak programok működését tudjuk felvázolni, hanem bármely más cselekedetet, vagy tervet.

14.5. BPEL Helló, Világ! - egy visszhang folyamat (zöld)

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.6. TeX UML

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15. fejezet

4. heť - „Hello, Chomsky!”

15.1. Encoding

Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezetes betűket!

link: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html

Megoldás forrása: https://github.com/BOPI98/prog2_feladatok/tree/master/Chomsky/Encoding

Tanulságok, tapasztalatok, magyarázat... : A feladat arrül szól, hogy a korábban már átvett mandelbrot halmaz nagyító programot egy olyan formában kell lefuttatni, hogy a fájlnévben és a forráskódban a változóknak is ékezetes karakterei vannak. A program forráskodja megtalálható a tankönyvtár.hu-n. Miután letöltöttük a programot és javac-vel compile-olni akarjuk tapasztalhatjuk, hogy rengeteg error-t kapunk az ékezeteknek köszönhetően. Ezt a problémár az -encoding kapcsolóval lehet megoldani, mégpedig a javac -encoding Cp1250 *.java vagy javac -encoding windows-1250 *.java paranccsal. Miután a forráskódunk lefordult készen is áll a futtatásra.

15.2. OOCWC lexer

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

link: https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src/carlexer.ll

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.3. l334d1c45(zöld)

Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: https://simple.wikipedia.org/wiki/Leet (Ha ez első részben nem tetted meg, akkor írasd ki és magyarázd meg a használt struktúratömb memóriafoglalását!)

link: https://github.com/BOPI98/prog2_feladatok/blob/master/Chomsky/l337/l337class.cpp

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... : Ez a feladat egy már régebben megoldott feladatból származtatik. A leet egy olyan program amely egy szövegből minden karakterére egy olyan karakterkonbinációt ír ki amely hasonlóan néz ki. Ebben a programban fordítás után a futtatás során bekér egy stringet, majd azokat a leet kicseréli egy random karakterre vagy karakterláncra a megadott tömbből.

15.4. Full screen(zöld)

Készítsünk egy teljes képernyős Java programot! Tipp: https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanitok-javat/ch03.html#labirintus_jatek

link: https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanitok-javat/ch03.html#labirintus_jatek

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... : A feladat egy teljes képernyős java alkalmazás készítése volt, melyehez kaptunk alapúl egy labirintus programot. A program letöltése után az encoding feladathoz hasonló eljárásokra volt szükség, hogy futtatható legyen. A futás során egy teljes képernyős alkalmazást kapunk, ami azt jelenti, hogy az egész képernyőt befedi a program. Az egyéni program egy panel osztályból és egy fő programból áll. A panel alapvetően elkészíti a programban látható oválisok alakját, színét. A fő program pedig a képernyő méretére igazodik, majd 10ms ütemben hozza létre az oválisokat.

15.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

Megoldás forrása: link: https://github.com/BOPI98/prog2_feladatok/blob/master/Chomsky/para/para6.cpp

Tanulságok, tapasztalatok, magyarázat... : A feladat egy opengl program változtatásáról szól. a program megtalálható a gitlab felhőn. A program letöltése után g++ para6.cpp -o para -lboost_system -lGL -lGLU -lglut parancsal lecompile-oljuk. A program alapvetően egy három dimenziós objektumot forgat, amelyen harom kocka van. A módisítás amit csináltam, az a háttér színe , amit glClearColor(0.0f,0.0f,0.0f,1.0f); függvénnyel tudunk megoldani. A második amit módosítottam, az a kocka színe volt fehérre, amit a drawPaRaCube függvénybe találhatunk meg. A harmadik a kocka kiválasztására szolgáló számok elcsúsztatása volt. Az utolsó pedig a forgatásnak a wads billenytűvel való irányítása volt.

15.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, még erősebb 3D-s hatás

link:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.7. Perceptron osztály

Dolgozzuk be egy külön projektbe a projekt Perceptron osztályát!

link: https://youtu.be/XpBnR31BRJY

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16. fejezet

5. heť - „Hello, Stroustrup!”

16.1. JDK osztályok

Írjunk olyan Boost C++ programot (indulj ki például a fénykardból) amely kilistázza a JDK összes osztályát (miután kicsomagoltuk az src.zip állományt, arra ráengedve)!

Megoldás videó:

Megoldás forrása: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf (16-22 foĺia)

Tanulságok, tapasztalatok, magyarázat...

16.2. Másoló-mozgató szemantika

Kódcsipeteken (copy és move ctor és assign) keresztül vesd össze a C++11 másoló és a mozgató szemantikáját, a mozgató konstruktort alapozd a mozgató értékadásra!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.3. Hibásan implementált RSA törése

Készítsünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: https://arato.inf.unideb.hu/batfai.n
(71-73 fólia) által készített titkos szövegen.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.4. Változó argumentumszámú ctor

Készítsünk olyan példát, amely egy képet tesz az alábbi projekt Perceptron osztályának bemenetére és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű „képet” adjon vissza. (Lásd még a 4 hét/Perceptron osztály feladatot is.)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.5. Összefoglaló

Az előző 4 feladat egyikéről írj egy 1 oldalas bemutató „”esszé szöveget!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

16.6. Általános

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

16.7. C

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

16.8. C++

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

16.9. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/-pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, https://groups.google.com/forum/#!forum/nemespor, az UDPROG tanulószoba, https://www.facebook.com/groups/udprog, a DEAC-Hackers előszoba, https://www.facebook.com/-groups/DEACHackers (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPROG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségben született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.