

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Dr. Bátfai Norbert

Copyright (C) 2019, Fazekas Márk János, nagyfazek06@gmail.com

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

<https://www.gnu.org/licenses/fdl.html>

Engedélyt adunk Önnek a jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Free Software Foundation által kiadott GNU FDL 1.3-as, vagy bármely azt követő verziójának feltételei alapján. Nincs Nem Változtatható szakasz, nincs Címlapszöveg, nincs Hátlapszöveg.

<http://gnu.hu/fdl.html>

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert Ács Fazekas, Márk	2019. május 9.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	5
2.3. Változók értékének felcserélése	7
2.4. Labdapattogás	7
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	7
2.6. Helló, Google!	7
2.7. 100 éves a Brun tétel	8
2.8. A Monty Hall probléma	8
3. Helló, Chomsky!	9
3.1. Decimálisból unárisba átváltó Turing gép	9
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	9
3.3. Hivatkozási nyelv	10
3.4. Saját lexikális elemző	10
3.5. l33t.1	11
3.6. A források olvasása	11
3.7. Logikus	12
3.8. Deklaráció	12

4. Helló, Caesar!	14
4.1. int *** háromszögmátrix	14
4.2. C EXOR titkosító	14
4.3. Java EXOR titkosító	14
4.4. C EXOR törő	15
4.5. Neurális OR, AND és EXOR kapu	15
4.6. Hiba-visszaterjesztéses perceptron	15
5. Helló, Mandelbrot!	16
5.1. A Mandelbrot halmaz	16
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	16
5.3. Biomorfok	17
5.4. A Mandelbrot halmaz CUDA megvalósítása	17
5.5. Mandelbrot nagyító és utazó C++ nyelven	17
5.6. Mandelbrot nagyító és utazó Java nyelven	17
6. Helló, Welch!	18
6.1. Első osztályom	18
6.2. LZW	18
6.3. Fabejárás	18
6.4. Tag a gyökér	19
6.5. Mutató a gyökér	19
6.6. Mozgató szemantika	19
7. Helló, Conway!	20
7.1. Hangyaszimulációk	20
7.2. Java életjáték	20
7.3. Qt C++ életjáték	20
7.4. BrainB Benchmark	21
8. Helló, Schwarzenegger!	22
8.1. Szoftmax Py MNIST	22
8.2. Mély MNIST	22
8.3. Minecraft-MALMÖ	22

9. Helló, Chaitin!	24
9.1. Iteratív és rekurzív faktoriális Lisp-ben	24
9.2. Gimp Scheme Script-fu: króm effekt	24
9.3. Gimp Scheme Script-fu: név mandala	24
10. Helló, Gutenberg!	25
10.1. Programozási alapfogalmak	25
10.2. Programozás bevezetés	27
10.3. Programozás	27
III. Második felvonás	29
11. Helló, Arroway!	31
11.1. A BPP algoritmus Java megvalósítása	31
11.2. Java osztályok a Pi-ben	31
IV. Irodalomjegyzék	32
11.3. Általános	33
11.4. C	33
11.5. C++	33
11.6. Lisp	33

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 dlatex 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
dlatex 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 'dlatex' 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 találsz 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.

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/vegtelena.c> <https://gitlab.com/nagyfazek06/programozas/blob/master/vegtelenc.c>

Hogy végtelen ciklust alkossunk, mindig igazzá kell tennünk a benne lévő feltételt (Kilépni a ctrl+c billentyű kombinációval lehet). Ha ez megvan az omp segítségével a magokat tudjuk terhelni.

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 végtelen 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 épülő Lefagy2 már nem tartalmaz feltételezett, 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.

Mivel egy ilyen program elméletben kivitelezhetetlen, ezért a gyakorlatban sem megvalósítható.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/csere.c>

Felveszünk 2 változót(`a` és `b`-t). Ezek értékét egy segéd változóval cserélhetjük fel. A segéd változó megkapja először az `a` értékét, majd az `a` megkapja a `b` értékét, majd a `b` megkapja a segéd változó értékét és készen is vagyunk.

2.4. Labdapattogás

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

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

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/ifbunce.c> <https://gitlab.com/nagyfazek06/programozas/blob/master/labdapattogas.c>

Tanulságok, tapasztalatok, magyarázat...

2.5. Szóhossz és a Linus Torvalds féle `BogoMIPS`

Írd 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://gitlab.com/nagyfazek06/programozas/blob/master/szohossz.c>

Egy egyszerű bitshifteléssel végig megyünk egy `int`-en aminek a mérete általában 32.

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/pagerank.c>

A Pagerank algoritmussal az oldalak fontosságát lehet megállapítani. Az algoritmus megvizsgálja az oldalakat majd eldönti melyik a leggyakrabban látogatott oldal. Böngészés során ennek az algoritmus által előállított rangsor segítségével találhatjuk meg a keresett oldalakat.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

A Monty Hall paradoxon egy Amerikai TV-adásának (Let's make a deal) utolsó feladatán alapul. A játékosnak 3 ajtó mögül kell választania amelyek mögött 1 autó és 2 kecske van. A játékos miután ajtót választott a átévezető (aki tudja hol mi van) kinyit egy ajtót (mindig olyat nyit ami mögött kecske van). Az ajtó kinyitása után a játévezető megadja a lehetőséget, hogy a játékos meggondolja a választását. Ekkor dönti el a játékos hogy vált e vagy sem és a döntés után a játévezető kinyitja a választott ajtót, s a játékos megkapja a nyereményét. A paradoxon kérdése, hogy megéri-e váltani.

3. fejezet

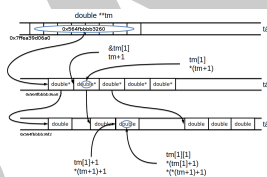
Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás



Az unáris számrendszerben az ábrázolás n darab egyforma karakter egymás utáni leírásával történik. A decimálisból unárisba váltás: egy számból folyamatosan 1-eket vonunk ki mindaddig míg 0 nem lesz, és tároljuk a levont egyeseket számát.

3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen

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

Megoldás videó:

Megoldás forrása: Első:

Szabályok:

$$S \rightarrow aBSc$$
$$S \rightarrow abc$$
$$Ba \rightarrow aB$$
$$Bb \rightarrow bb$$

Levezetés: $S \rightarrow aBSc \rightarrow aBaBabccc \rightarrow aaBBabccc \rightarrow aaBaBbccc \leftarrow$
 $\rightarrow aaaBBbccc \rightarrow aaBbbcc \rightarrow aaabbbccc$

Második:

Szabályok:

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

Levezetés:

```
S -> aXbc -> abXc -> abYbcc -> aYbbcc -> aaXbbcc -> aabXbcc ↔
    -> aabbXcc -> aabbYbcc -> aabYbbcc -> aaYbbbcc -> ↔
    aabbbccc
```

Noam Chomsky alkotta meg a generatív nyelvtan elméletét illetve ő dolgozta ki a Chomsky-hierarchiát. A grammatikának három típusa van: a környezet független, a szabályos és a generatív nyelvtan. A környezetfüggetlen nyelvtanban a szabályok megadása esetén a bal oldalt nem terminális változó, a jobb oldalt terminális változók állnak.

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://gitlab.com/nagyfazek06/programozas/blob/master/hivatkozas.c>

A Backus-Naur-forma a különböző nyelvek egyik lehetséges leírási módszere. Ezt a leírási módszert John Backus hozta létre, eredetileg az ALGOL programozási nyelvhez. Azóta már a legtöbb programozási nyelv szintaxisát BNF-ben adják meg, illetve természetes leírásához is használják alkalmanként. Peter Naur egyszerűsítette le a leírási módszert, ezért Donald Knuth javaslatára Naur neve is belekerült a leírási módszer megnevezésébe. Feladat volt még olyan C programot írni, amely egyes nyelvi szabvánnyal, jelen esetben a C89-es szabvánnyal, nem fordul le, míg például a C99-es szabvánnyal már igen. A C89-es szabvány például még nem engedte a for ciklusban a ciklusfejben történő ciklusváltozó deklarálását. Ezt szemlélteti a fenti kód is, ugyanis a -std=c89 kapcsolót használva hibát jelez a fordító.

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/realnumber.l>

Az lex fájl első részében, amit a `%{...%}` jelöl, jelzem a fordítónak, hogy az `stdio.h` függvénykönyvtárat szeretném használni az `stdout`-ra való íráshoz, majd létrehozok egy számlálót, ami a lexer által észlelt számok darabszámát fogja tárolni. A második részben, amit a `%%...%%` jelöl, megadtam a lexernek, hogy bizonyos mintákra hogyan reagáljon. A `[[digit:]]+` azt jelenti, hogy legalább egy számjegy egymás után. Ha legalább egy számjegyet talál egymás után, abban az esetben eggyel növeli a számlálót. Ezután megadtam, hogy bármilyen más minta esetében, effektíve ne csináljon semmit. Majd az utolsó részben, amely már nincs külön jelölve, a `main` függvényben meghívjuk a `yylex` függvényt, amely meghívja magát a lexert, amely végigfutja bájtonként a bemenetet. Ha a lexer futása véget ért, kiíratom a számláló értékét. Majd a `return 0`-val jelzi az operációs rendszer felé, hogy a program futása véget ért.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/l337d1c7.1>

Mi az a leet? A leet lényege hogy a kerektereket ASCII karakterekkel helyettesítse. Ennek célja a titkosítás.

3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezeslo)==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 `jelkezeslo` 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 meggyőződésre, elkapja valamelyiket esetleg a splint vagy a frama?

i.

```
if(signal(SIGINT, SIG_IGN)!=SIG_IGN)
    signal(SIGINT, jelkezeslo);
```

ii.

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

iii.

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

iv.

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

v.

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

vi.

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

vii.

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

viii.

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

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/jelkezelok.c>

Megoldás videó:

Ebben a feladatban olyan végtelen ciklust készítünk amiből ctrl+c -vel sem láphetünk ki, ugyanis a SIG-NINT jelkezelő megakadályozza.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

```
 $\$ (\backslash \text{forall } x \backslash \text{exists } y ((x < y) \backslash \text{wedge} (y \text{ \textit{prím}})) \backslash \text{wedge} (Ssy \text{ \textit{prím}})) \leftrightarrow$   
 $) \$$ 
```

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

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

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

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

Az első formula természetes nyelven: Minden x -re létezik olyan y , hogy x kisebb, mint y , és y prím. Tehát: Minden számnál létezik nagyobb prímszám. A második formula természetes nyelven: Minden x -re létezik olyan y , hogy x kisebb, mint y , y prím és y rákövetkezőjének rákövetkezője is prím. Tehát: Minden számnál léteznek nagyobb ikerprímek. A harmadik formula természetes nyelven: Létezik olyan y , hogy minden x -re igaz, hogy ha x prím, akkor x kisebb, mint y . Tehát: Minden prímszámmra igaz, hogy létezik tőle nagyobb szám. 3.8 Deklaráció FT A negyedik formula természetes nyelven: Létezik olyan y , hogy minden x -re igaz, hogy ha y kisebb, mint x , akkor x nem prím. Tehát: Létezik olyan szám, amelytől nem létezik kisebb prímszám.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/tm.c>

A program egy double-öket tartalmazó alsó háromszögmátrixnak helyet foglalok a memóriában ,majd k=0-tól a mátrix minden elemének értékül adtuk k-t. Ezután a mátrixot standart outputra kiírjuk. Következőnek felszabadítjuk a memóriacímeket amelyeket a pointerek foglalnak le, majd a "return 0"-val jelezzük az operációs rendszer felé, hogy a program hibátlanul lefutott.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/exor.c>

Ez a program a legelején megnézi, hogy a kapott argumentumok száma több, mint kettő. Erre azért van szükség, mert a kódolandó szövegfájl nevét és a titkosításhoz használt kulcsot parancssori argumentumként kapja meg a program. Ha a feltétel teljesül, tovább fut a program, ha nem, akkor kiírja standard outputra a program helyes használatának módját. Ezután az igaz ágon belül megnyitjuk olvasásra a kódolandó szöveget tartalmazó fájlt. Ha a fájl megnyitása nem sikerült, hibaüzenettel kilép a program. Ha sikerült, akkor karakterenként beolvassuk, a kulcs megfelelő karakterével "össze-exorozzuk", majd kiíratjuk standard outputra. Ha végezett a beolvasással, akkor a tiszta szöveget tartalmazó fájlt bezárjuk, majd kilép és jelzi az operációs rendszer felé, hogy a program futása hiba nélkül véget ér.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/exor.java>

Ez a program az előző feladat megoldásának java átirata, amely annyival különbözik az előzőtől, hogy itt a tiszta szöveget tartalmazó fájl neve tiszta.txt, amely egy mappában van a programmal, illetve a kulcsot a standard inputról kéri be, nem parancssori argumentumokként.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/toro.cs>

Ez a program megpróbálja feltörni az előző feladatban titkosított szöveget.

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Ebben a feladatban a cél egy olyan neurális háló létrehozása és tanítása, amely az egyszerű logikai műveletek elvégzésére képes. Ez a kód igazából négy kis eltéréssel ismétlődő részből áll. Minden részben lényegében ugyanaz történik, egyedül a logikai művelet változik, amelyre feltanítjuk a neurális hálót. Ez alól kivétel az utolsó, amiről lentebb szó lesz.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/perceptron.cpp>

A program a kód elején vizsgálja, hogy a parancssori argumentumok száma kettő-e. Erre azért van szükség, mert az bemenetként szolgáló png fájl nevét parancssori argumentumként kapja meg a program. Ezután a png++ függvénykönyvtár segítségével beolvassa a bementi fájlt. Ezután létrehoz egy perceptront 4 rétegű neurális hálóval, amelynek neuron száma sorra 3, a kép pixeleinek száma, 256, 1. Majd létrehoz egy dinamikusan foglalt a kép pixeleinek megfelelő számosságú double tömböt, amelybe belemásolja a kép minden pixelének vörös értékét. Ezután meghívja a perceptront, hogy dolgozza fel a képet. A program végén felszabadítja a pointerek által foglalt memóriát, majd kilép.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/mandelbrot.cpp>

A program elején létrehozunk a precompiler számára pár változót, amelyek a kimeneti kép méretét határozzák meg. A GeneratePNG függvény paraméterként megkapja a program által generált kép adatait pixelenként a tomb nevű int típusú $N \times M$ -es mátrixban tárolva. A függvényen belül létrehozunk egy $N \times M$ pixeles PNG kiterjesztésű képfájlt, amibe az egymásba ágyazott for ciklus beletölti az adott pixelre vonatkozó adatokat, majd a for ciklus után kiírja a képfájlt. A main függvényen belül létrehozuk az $N \times M$ -es mátrixot, amelyben tároljuk a kép adatait, beállítjuk a komplex számsíkon való lépegetés közét a "dx" és "dy" változóban, majd létrehozunk három Komplex típusú változót, amely a komplex számokat fogja tárolni. Ezután belép a program a for ciklusba ahol a komplex számokkal lépked a megadott tartományban a megadott lépésközzel, majd meghatározza a while ciklus a kép megfelelő pixelének színét annak függvényében, hogy a while ciklus fejében megadott formula hány iteráció alatt lesz nagyobb vagy egyenlő, mint négy. Miután a for ciklus bejárta a komplex számsík megadott tartományát, meghívja a GeneratePNG függvényt. (Segített Petrus Tamás József)

5.2. A Mandelbrot halmaz a std::complex osztállyal

Megoldás videó:

Megoldás forrása: https://gitlab.com/nagyfazek06/programozas/blob/master/mandel_complex.cpp

```
struct Komplex{  
    double re, im;  
};
```

Változás az előző programhoz képest hogy c++ beépített complex osztályát használva lépegetünk a komplex számsíkon.

5.3. Biomorfok

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

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

A biomorfok olyan alakzatok, amelyek ránézésre élő organizmusok is lehetnének, viszont nem muszáj természetes eredetűnek lennie az alakzatnak. Ez a program nagyon hasonlít az előzőhöz, ugyanis ennek az alapja a Mandelbrot-halmaz. A legjelentősebb eltérés az előző programhoz képest, hogy itt más a megadott formula a pixelek színeinek számításánál.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása: https://gitlab.com/nagyfazek06/programozas/blob/master/mandelbrot_nagyito.cpp

Megoldás videó:

A feladat megoldásához SFML-t használtam. A feladatban a mandelbrot halmaz meghatározására az 5.2-es feladat megoldását vettem alapul. A compute függvény határozza meg minden pixel színét, ugyanazon módszerrel, mint az 5.2-es feladatban. A következő néhány függvény segédfüggvény a nagyításhoz illetve a halmaz feltérképezéséhez. A main függvényben létrehozom a grafikus megjelenítéshez szükséges objektumokat, változókat. A while cikluson belül először eseménykezelés található, amely azért felelős, hogy mi történjen, ha a felhasználó a képernyőre kattint, illetve ha a egér görgőjével görget. Majd meghívom a compute függvényt, ezután pixelenként kirajzoltatom a kiszámolt mandelbrot-halmazt. (Segített Petrus Tamás József)

5.6. Mandelbrot nagyító és utazó Java nyelven

<https://gitlab.com/nagyfazek06/programozas/blob/master/mandelb.java>

Az osztály konstruktorában létrehozuk a GUI-t, illetve megadjuk a paramétereit, létrehozuk a kontroll objektumokat és a gombokat, amelyekkel változtatható a vizsgált komplex szám tartomány. A plotPoints eljárás felelős a vizsgált tartomány bejárásáért. A actionPerformed eljárás felelős azért, hogy az interakciókat lehetővé tegye azáltal, hogy megmondja mi történjen, ha a felhasználó egy gombra nyom.

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 elspájzold és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltérve kiszámolt szám.

Megoldás videó:

Megoldás forrása: c++ : <https://gitlab.com/nagyfazek06/programozas/blob/master/osztaly.cpp> java: <https://gitlab.com/nagyfazek06/programozas/blob/master/osztaly.java>

Létrehozuk a polargen nevű osztályt ahol a konstruktorba azt adjuk hogy nincs szám. A "kovetkezo" függvény megnézi, hogy van e tárolt szám. Ha nincs generálunk 2darabot. Az egyiket eltárolja majd a logikai változó hamis értéket vesz fel és a másik generált számmal tér vissza. Az objektum orientált programozás természetes, hisz az emberek a világon mindent objectumként látnak amelyeknek adataik vannak.

6.2. LZW

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/binfa.c>

A program elején létrehoztunk egy Node nevű struktúrát, amely az LZW bináris fában lévő csomópontokat reprezentálja. Ennek van egy "char" típusú változója, illetve két önmagára mutató mutató. Az egyik a nullás gyerekre mutat, a másik gyerekre. A "create_empty" függvény inicializálja a fát egy kitüntetett elemmel, amely "/" -el van jelölve. Az "insert_tree" hozza létre a LZW binfa felépítését. A "main" függvényben a fát feltöltjük x elemmel és ezután végigjárjuk a fát.

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/binfa.c>

Preorder: először feldolgozzuk a fa gyökerét majd bejárjuk a fa bal oldalát és ennek végeztével a fa jobb oldalát. Postorder: először bejárjuk a fa bal oldalát majd a jobb oldalát és végezetül feldolgozzuk a gyökeret.

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/binfa.cpp>

A bináris fát kezelő függvényeket és eljárásokat a Binfa osztályba rendezzük, illetve a Binfa osztály privát részébe a Node struktúra kerül. Az osztályon belül a balra bitshift operátort túlterheljük, ami mostmár a bináris fa építését látja el.

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://gitlab.com/nagyfazek06/programozas/blob/master/binfa6_5.cpp

Ez a megoldás az előző feladat megoldásának egy módosítása. A különbség, hogy ebben a gyökérelemre is már egy mutató mutat, azért a Binfa konstruktorában létre kell hozni a gyökérobjektumot. Ahol a program eddig a gyökérelem referenciáját adta át függvénynek, vagy a faépítő eljárásban, ott mostmár a gyökérelemet kell átadni és nem a referenciáját.

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://gitlab.com/nagyfazek06/programozas/blob/master/mozgato.cpp>

Ez a megoldás a Tag a gyökér (6.4) feladat kiegészítése másoló és mozgató szemantikával. A másoló szemantika lényege, hogy az értékül kapott bináris fát értékül adja az eredeti fának, minden érték lemásolásával. A mozgató szemantika pedig úgy működik, hogy az eredeti bináris fa gyökerét kicseréli az értékül kapott fa gyökerével, és az értékül kapott fa gyökerének gyermekeit nullpointerre állítja.

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása: <https://github.com/nagyfazek/antfarm>

Ennek a programnak a célja hogy a hangyák utakat hozzanak létre amellyen közlekednek, szimulálja őket. Ha egy út már létezik a hangyák azon közlekednek.

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/ant.java>

A program elején importáljuk a szükséges csomagokat melyek kezelik a az out- illetve inputot. A `game_of_life` egyetlen publikus osztályába található a `main`. Egy `RenderArea` típusú változót hozunk létre, amelyet majd a `game_of_life` konstruktorában inicializálunk. A konstruktor emellett még beállítja az ablak címét és egyéb tulajdonságait.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://github.com/nagyfazek/életgame>

A program 2 osztályból áll: az első a hálót rajzolja mely a sejteket jelképezi és a második a sejtek helyét és állapotát tárolja a hálóban. Az osztályokon kívül a függvények a sejteket az élet szabályai alapján vizsgál.

7.4. BrainB Benchmark

Megoldás videó:

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

Ez a program azt vizsgálja hogy mennyire vagyunk képesek odafigyelni a játékbeli karakterünkre. Ezen kívül azt is vizsgálja hogy ha szemelől tévesztjük karakterünket akkor mennyi idő alatt találjuk meg újra. Ez az online játékoknál fontos főleg a MOBA típusuaknál. Ebbe a műfajba elég gyakori, hogy nagyobb harcoknál a zűrzavar közepén elvesztjük szemelől a karakterünket így nem tudjuk a játszma kimenetelét befolyásolni ami akár annak elvesztéséhez vezethet.

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/exar
https://progater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

A program elején importáljuk a tensorflow modult, amely segítségével betöltjük az MNIST adatbázis adatait. Az `x_train`, `y_train`, `x_test`, `y_test` változóiban eltároljuk az MNIST adatbázisban található képeket az ehhez tartozó címkéket. Ezután konvertáljuk az `x_train` és `x_test` tömbök értékeit float típusúra, majd normalizáljuk az értékeket. Importálunk még hat modult, amelyek a neurális háló felépítéséhez szükségesek. Ezután létrehozunk a neurális háló modelljét, majd feltanítjuk a hálót. Ezután pedig kiértékeljük a neurális háló "tudását" a teszt képekkel és címkékkel.

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/deep.py>

A program elején importáljuk a szükséges modulokat. Létrehozunk 4 függvényt, amelyek a programot rövidítik, ezután létrehozunk a neurális hálót. Ebben a programban 5 rétegű a neurális háló. A rétegek megadása után megadjuk a módot hogy hogyan szeretnénk kiértékelteni a bemenetet, majd megadjuk a megfelelő algoritmust amivel feltanítjuk a hálót. Ezután következik a tanítás, illetve minden századik iterációban leteszteljük a hálót.

8.3. Minecraft-MALMÖ

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

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/malmo.py>

A Minecraft-MALMÖ projektet azért hozta létre a Microsoft, hogy a Minecrafton belül lehessen mesterséges intelligenciákat létrehozni. Az programban alapvetően az a cél Steve-nek hogy ne akadjon el 5 percen keresztül, miközben folyamatosan megy. Megvizsgálja az előtte lévő 12 blokkot, és bizonyos feltételek teljesülése esetén ugrik, elfordul és halad más irányban. A program megírásához felhasználtam a példaprogramot. A programban először importáljuk a szükséges modulokat, illetve a MALMÖ API-t. A `restart_minecraft` függvény arra szolgál, hogy a mission lefutása után a Minecraft új missiont futtasson. A `run` függvényben található a mission pontos megadása. A `missionXML` string-ben van megadva a világ generálásának szabályai. Ezután megadjuk, hogy 300 másodpercig fusson a mission, illetve, hogy 640x480-as felbontással fusson a Minecraft. Az első while ciklus vár addig, amíg a kliens készen áll a program általi irányítást fogadni. A második while ciklusban van Steve vezérlése. A ciklus elején az első két elágazás arra szolgál, hogy Steve ne folyamatosan ugráljon hanem át tudjak ugrani vagy fel tudjon ugrani az akadályra. A harmadik elágazás az elfordulásra szolgál. Az ezt követő elágazás eldönti hogy merre néz Steve. Erre azért szükséges hogy a környező blokkokat lehessen vizsgálni. A `blocks` nevű kétdimenziós tömbbe töltöm bele a vizsgált blokkok nevét. Ezután a program az alapján, hogy Steve merre néz, megvizsgálja, hogy mit csináljon.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása: <https://gitlab.com/nagyfazek06/programozas/blob/master/fact.lisp?fbclid=IwAR0fvQjfm4nkgNNxOFeY4DNwT7EiEuZdg4Soyk7g> <https://gitlab.com/nagyfazek06/programozas/blob/master/recursive.lisp?fbclid=IwAR0fvQjfm4nkgNNxOFeY4DNwT7EiEuZdg4Soyk7g>

Iteratív: A program elején található a fact függvény ami kiszámolja a faktoriális értéket. Létrehoz egy változót sum névvel, amit megszorozok x-xel amit ezután csökkentek 1-el. A függvény létrehozását követően kérünk inputról egy számot, majd lefutatjuk erre a számra a fact függvényt. Rekurzív: A program elején létrehozuk a factorial függvény ami rekurzív módon számolja ki a faktoriális értéket. A függvényben megvizsgálom x-et hogy megegyezik-e 1-el. Ha igen visszatérési értéke 1 lesz, ha nagyobb akkor egyenlő értékű lesz x factorial-al.

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...

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...

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

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

A programozási nyelveknek három szintje van: a gépi nyelv amelyek már a processzor nyelvére vannak lefordítva, az assembly szintű nyelv, azaz gépközei nyelvek és a magas szintű nyelvek, mint például a java és C++. Minden forrás szöveget megfelelő kóddá kell alakítani, mivel minden processzor saját gépi nyelvvel rendelkezik. Erre két megoldás létezik: azinterpreteres és a fordítóprogramos. Az utóbbi megoldás a forrásszöveget lefordítja, így válik futtathatóvá, míg az előbbi esetében az interpreteres soronként halad végig és olvasási sorrendben hajtja végig az utasításokat. A programozási nyelveknek van saját hivatkozási szabványa. Ebben vannak definiálva a szintaktikai és szemantikai szabályok, legtöbbször angolul. Vannak implementációk, amelyek operációs rendszereken interpreter megvalósítást jelent. Ezek nem kompatibilisek egymással. Létezhet egy operációs rendszeren több implementáció is, ezek sem mindig kompatibilisek egymással. Manapság a programozáshoz többnyire IDE-ket használunk, amelyek grafikus programok, amelyekbe van beépített szövegszerkesztő, fordító és futtatórendszer.

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

Az adatabsztrakció első formája az adattípus. Az adattípus rendelkezik névvel, ami azonosítja a típust. Léteznek nem típusos és típusos programozási nyelvek. A típusosok engedik meg, hogy a programozó megadja a változók típusát, például a Java. A nem típusosok automatikusan adják meg a változó típusát, például a Python. Adattípusoknak két csoportja van, az egyszerű és az összetett. Az egyszerű adattípusokat nem lehet tovább bontani, míg az összetett típusokat lehet, például a struktúrákat.

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

A nevesített konstansok azt a célt szolgálják a programokban, hogy olyan nevet adjunk a konstansoknak, amely jelképezi a típusát és értékét, illetve másik célja, hogy többszöri használat esetén csak a megadásnál kelljen változtatni az értékét. Ezeket mindig definiálni kell.

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

A változónak 4 komponense van: a név, az attribútumok, a cím és az érték. A név az egy azonosító, amit a másik három komponens egy névhez rendeljük hozzá. A típus a legfőbb attribútum, mely a változó által felvett értéket határolja be. A változóhoz deklarációk segítségével rendelődnek az attribútumok. A deklarációnak több különböző fajtát ismerünk: Explicit deklaráció, Implicit deklaráció, Automatikus deklaráció. A

változó értékének a helyét a változó címe határozza meg. A címrendelésnek 3 fajtáját ismerjük: a Statikus, a Dinamikus, és a programozó által vezérelt kiosztást. A változó értékét megadhatjuk értékadó utasítással vagy kezdőérték adással.

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

C-ben az aritmetikai típusok az egyszerű típusok és a származtatottak az összetett típusok. A belső kódok alkotják a karakter típus elemeit. Logikai típus nincs ezért a hamis az int 0 és az igaz pedig az int 1. A void tartománya üres. A felsorolás típusok nem fedhetik egymást. Ugyanazt az értéket hozzárendelhetjük különböző elemekhez.

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

A kifejezések szintaktikai eszközök. A kifejezések operandusokból, operátorokból és kerek zárójelekből állnak. Létezik egyoperandusú, kétoperandusú és háromoperandusú operátor, melyek attól függnak, hogy hány operandussal végzi a műveletet az operátor. A kifejezéseknek 3 alakja lehet: a prefix, az infix és a postfix. Kifejezés kiértékelésének nevezzük azt a folyamatot mikor a kifejezés értéke és a típusa meghatározódik. A kifejezések két típusa: a típusegyenértékűség és a típuskényszerítés.

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

Az utasítások megalkotják a programok egységeit. Két csoportjuk van: a deklarációs utasítások illetve végrehajtó utasítások. A deklarációs utasítások a fordítóprogramnak szólnak. A végrehajtó utasításokból a fordító generálja a kódot. A végrehajtó utasításokat csoportosíthatjuk többféle képpen: értékadó utasítás, üres utasítás, ugró utasítás, elágaztató utasítás, ciklusszervező utasítás, hívó utasítás, vezérlésátadó utasítás, I/O utasítás, egyéb utasítás. Vezérlési szerkezetet megvalósító utasítások a következők: ugró utasítás, elágaztató utasítás, ciklusszervező utasítás, hívó utasítás, vezérlésátadó utasítás.

VII. heti előadás (78-84. oldal):

A paraméterátadásnak többféle képpen is történhet, ezek nyelvfüggőek. Történhet érték szerint, mint a C-ben például. Ekkor az aktuális paraméter értékét kapja a formális paraméter. Ilyenkor a függvényben nem lehet megváltoztatni a paraméter értékét. Lehet címszerinti a paraméterátadás. Ekkor a formális paraméter címe kapja az aktuális paraméter címét. Itt a függvényben meg lehet változtatni az aktuális paraméter értékét. Eredmény szerinti átadás esetén, a formális paraméter szintén megkapja az aktuális paramétert, de nem használja, hanem beletölti az adatokat. Létezik még érték-eredmény szerinti, ekkor is másolódik a cím illetve használja is az adatokat és a függvény végén belemásolja az adatokat.

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

Az absztrakt adattípus olyan adattípus, amely megvalósítja a bezárást vagy információ rejtést. Az ilyen típusú programozás művelethez a specifikációi által meghatározott interfészen keresztül férhetük hozzá ezért az értékeket véletlenül vagy szándékosan nem ronthatjuk el.

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

A generikus programozás az újrafelhasználhatóság és a procedurális absztrakció eszköze. Lényege: Megadunk egy paraméterezhető forrásszövegmintát ami fordítási időben lesz kezelve majd kezelve. A mintaszöveg segítségével előállítható egy lefordítható konkrét szöveg. Az újrafelhasználás alatt azt értjük, hogy egy mintaszövegből bármennyi szöveg generálható. A generikus formális paramétereinek száma mindig fix. A paraméterkiértékelésnél a kötés az alapértelmezett, de alkalmazható a név szerinti kötés is.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

V.heti előadás (Vezérlési szerkezetek című fejezet):

Egy nyelv vezérlésátadó utasításai a műveletek végrehajtási sorrendjét határozzák meg. A C nyelvben a ; az utasításlezáró jel. A kapcsos zárójelekkel deklarációk és utasítások csoportját fogjuk össze egyetlen összetett blokba. Az "if-else" utasítás döntés kifejezésére használjuk, az utasítás először kiértékeli a kifejezést, és ha igaz, akkor az első utasítást hajtja végre de ha a kifejezés értéke viszont nem igaz, akkor az "else" ág fut le. Általános szabály, hogy az "else" mindig a legközelebbi "if"-hez tartozik. A "switch" utasítás is a többirányú programelágazás egyik eszköze, összehasonlítja egy kifejezés értékét több egész értékű állandó kifejezés értékével és annak megfelelő utasítást hajtja végre. A "while - for" szerkezet először kiértékeli a kifejezést, ha ennek az értéke nem nulla, akkor végrehajtja az utasítást. Ez addig ismétlődik, amíg nulla nem lesz a kifejezés. A "do - while" szerkezet először végrehajtja az utasítást, és csak utána értékeli ki a kifejezést. Ha a kifejezés értéke igaz, akkor az utasítást újból végrehajtja. Ez addig ismétlődik, amíg a kifejezés értéke hamis lesz. A "break" lehető teszi, hogy elhagyjuk a ciklusokat idő előtt. A "goto" utasítás előnyös, ha ki akarunk lépni egy több szinten egymásba ágyazott ciklusból.

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

Az utasítások a leírásuk sorrendjében hajtódnak végre, általános a szintaktikai leírásuk, és számos csoportba sorolhatók: Címkezett utasítások, mint például a "case" és "default" címkéi a "switch" utasítással használhatók. A címke egy azonosító nélküli deklarált azonosítóból áll. Kifejezésutasítások, az utasítások(kifejezésutasítás, értékadás, függvényhívás) többsége ilyen. Összetett utasítás, több utasítást egyetlen utasításként kezeli, ez a fordításhoz szükséges, mivel sok fordítóprogram csak egyetlen utasítást fogad el. Kiválasztott utasítások, minden esetben a lehetséges végrehajtási sorrendek egyikét választják ki(if, if-else, switch). Iterációs utasítások, egy ciklust határoznak meg(while, do-while, for). Vezérlésátadó utasítások, vezérlés feltétel nélküli átadására alkalmasak(goto, continue, break, return).

10.3. Programozás

[BMECPP]

V.heti előadás (1.-16.):

A C++ a C-nek a továbbfejlesztése. A C++ sok problémára biztonságosabb, és kényelmesebb megoldást kínál, mint a C. C-ben üres paraméterlistával definiálunk, akkor az tetszőleges számú paraméterrel hívható. A C++-ban azonban az üres paraméterlista egy "void" paraméter megadásával ekvivalens. C nyelvben is létezik több bájtos sztring. C++-ban minden olyan helyen állhat változódeklaráció, ahol utasítás állhat. A C nyelvben a neve azonosít egy függvényt, C++-ban viszont a függvényeket a nevük, és az argumentumlistájuk azonosítja. Míg a C nyelv úgy hivatkozik egy függvényre a linker szintjén, hogy egy aláhúzást tesz a függvénynevek elé, addig a C++ az egyes fordítókra bízta a névferdítés implementálását. Cím szerinti paraméterátadás, ha a változó címét adjuk át, ebben az esetben nem tudjuk megváltoztatni úgy a változót, hogy az értéke megmaradjon. Az érték szerinti paraméterátadásnál viszont, készül másolat a változóról, így végezhetünk műveleteket úgy, hogy a változó értékét nem befolyásoljuk. A C++ referenciatípus bevezetése feleslegessé teszi a pointerok cím szerinti paraméterátadását.

VI. heti előadás (17-58.):

Ez a fejezet a C++ osztályairól szól. Az objektum orientált programozás alapelve, hogy a probléma megoldását segítse azzal, hogy az emberi gondolkodáshoz közelebb hozza a programozást az osztályok és objektumok bevezetésével. Az egységbe záras jelenti azt, hogy az összetartozó változók és függvények egy egységben legyen, ezek lesznek az adattagok és a tagfüggvények. Adatrejtés a private és protected adattagok és tagfüggvények bevezetésével jött létre. Az adatrejtés célja, hogy az osztály egyes tagjait ne lehessen kívülről elérni. A konstruktor szerepe, hogy lefusson, amikor létrejön az objektum, ezáltal akár inicializálva az adattagokat. A destruktorként célja, hogy lefusson, amikor az objektum megsemmisül, ezáltal akár felszabadítva a dinamikus adattagokat. A dinamikus adattagok osztályon belüli pointerok, amelyeket futásidőben hozzuk létre dinamikus memóiafoglalással, ezért ezeket, ha már nincs rájuk szükség, de legkésőbb a destruktorként fel kell szabadítani. A friend osztályok, illetve függvények olyan osztályok, illetve függvények, amelyek ugyan nem tagjai az osztálynak, viszont hozzáférnek azok private tagjaihoz. A tagváltozók inicializálása történhet a konstruktoron belül, illetve tagfüggvénnyel, vagy külső függvénnyel is. A statikus tagok azzal a tulajdonsággal rendelkeznek, hogy nem kell az osztályt példányosítani, hogy használni tudjuk. Az osztályok tartalmazhatnak beágyazott definíciókat, amelyek lehetnek enumerációk, struktúrák vagy akár osztályok is.

VII. heti előadás (93-96.):

Ez a fejezet az operátorokról és azok túlterheléséről szól C++ nyelvben. Az operátorok alapértelmezés szerint az argumentumain hadjtanak végre műveletet, amelyek visszatérési értékével tudunk dolgozni. C++-ban nem lehet új operátorokat létrehozni, azonban majdnem mindet túl lehet terhelni. A túlterhelés célja, hogy bizonyos operátorokat más célra használhassunk, mint az eredeti célja, a programozás megkönnyítése céljából, mint például a bitshift operátor túlterhelése az alapértelmezett IO objektumoknál.

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.

DRAFT

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

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

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

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