# Univerzális programozás

## Így neveld a programozód!



Ed. BHAX, DEBRECEN,
2019. március 25., v. 0.1.3

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>Univerzális programozás | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Bátfai, Norbert Ã©s Ratku, Dániel | 2019. március 26. | |

| 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 |
| 0.0.5 | 2019-03-01 | A Helló, Turing csokor megkezdése és a dokumentum formájának a személyre szabása és a saját repository-ba való feltöltés. | ratku.dani |
| 0.0.6 | 2019-03-04 | A Helló, Turing csokor nyolc feladatából az első öt feladatának befejezése. | ratku.dani |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| 0.0.7 | 2019-03-05 | A Helló, Turing csokor befejezése és a Helló, Chomsky feladatcsokor átfutása. | ratku.dani |
| 0.0.8 | 2019-03-08 | A Helló, Chomsky feladat csokor megkezdése, és a github frissítése. | ratku.dani |
| 0.0.9 | 2019-03-11 | A Helló, Chomsky csokor befejezése, és a teljes feladatsor feltöltése githubra. | ratku.dani |
| 0.1.0 | 2019-03-15 | A Helló, Caesar feladatok elkezdése. | ratku.dani |
| 0.1.1 | 2019-03-17 | A Helló, Caesar csokor felének a kidolgozása. | ratku.dani |
| 0.1.2 | 2019-03-18 | A Helló, Caesar feladatcsokor majdnem teljes elkészítése. | ratku.dani |
| 0.1.3 | 2019-03-25 | A Helló, Mandelbrot csokor kötelezőinek elkészítése. | ratku.dani |

# Ajánlás

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

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

# Tartalomjegyzék

# Előszó

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

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

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

## Hogyan forgasd

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

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

### Milyen nyelven nyomjuk?

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

### Hogyan nyomjuk?

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

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

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

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

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

# I. rész

# Bevezetés

# 1. fejezet

# Vízió

## 1.1. Mi a programozás?

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

## 1.2. Milyen doksikat olvassak el?

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

## 1.3. Milyen filmeket nézzek meg?

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

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

# II. rész

# Tematikus feladatok

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

# 2. fejezet

# Helló, Turing!

## 2.1. Végtelen ciklus

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

Megoldás forrása:

Egy magot altató program.

Egy magot 100%-ban kihasználó program..

Minden magot 100%-ban kihasználó program.

A feladat során, az egy magot 100%-osan kihasználó programban, egy olyan do ciklust használtam, amely sosem éri el az abban megadott értéket, ezáltal a végtelenségig futva terheli a processzort. Az összes magot 100%-ban leterhelő program során, meg kellett hívnunk az omp könyvtárat, amellyel a "#pragma omp parallel"-t, amivel elérjük, hogy párhuzamosan terheljük a processzor összes szálát. Emellett egy olyan paraméterek nélküli for ciklust, amellyel megkapjuk a kívánt kihasználtsági szintet. A magot altató programhoz meg kellett hívni az unistd.h nevezetű könyvtárat, amivel tudjuk már használni a nem standard sleep függvényt, ami úgy működik, hogy a bele implantált érték ideéig leállítja a processzor egy szálát. Az egész feladatot végtelen ciklusokkal oldottam meg.

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

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

Megoldás videó:

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

```
Program T100
{

  boolean Lefagy(Program P)
```

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

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

```
}
```

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

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

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

(A feladat megoldásában Pankotai Kristóf segített) Egy olyan szoftvert kell megírni, ami el tudja dönteni egy adott különböző programról, hogy az lefagy-e, azaz a végtelenségig tud-e futni. Ezt nem lehet elkészíteni, mivel olyan programot nem lehet írni, ami egy másik program végtelenségét vizsgálva megelőzné azt, és annak a végére érkezve eldönthetné, hogy ez egy befagyhatatlan program-e a végtelensége miatt.

## 2.3. Változók értékének felcserélése

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

Megoldás forrása:

Változókat felcserélő program.

(A feladat megoldásában Pankotai Kristóf segített)Ennek a feladatnak a megoldásához be kell kérni két számot, amelyet a megadott számokkal különböző műveleteket elvégezve megkapjuk ezeknek a sorrendben felcserélt változatát. A feladatban három művelet szerepel, amelyben a változókat adjuk és vonjuk ki egymásból úgy, hogy a kapott eredmény a feladatnak megfelelő módon jelenjen meg.

## 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 forrása:

Labdapattogtató feladat megoldása.

Labdapattogtató feladat if nélkül.

Ennek a feladatnak a megoldásához meg kell hívnunk az unistd.h és a curses.h könyvtárat. A void azért kell bele, mert nem kér be semmit a program. Először is inícializálnunk kell az ablakot, amelynél figyelembe vesszük a felhasználó által kihúzott méretet. Deklaráljuk a szükséges változókat, például a kiinduló értéket és a haladás irányát. Majd egy végtelen ciklusban az ablak méretének a változását is figyelembe véve vetítjük ki a labda pattogását. Ezt folyamatosan frissítjük és a usleep-et használva megadva a sebességét

halad a "labda". A pattogás menetét, azaz irányát if elágazásokkal határozzuk meg, szám szerint néggyel, amely tartalmazza a fel, le, jobbra, és a balra való irányokat.

Az if elágazások nélküli feladatnál általunk előre megadott ablak, azaz pályaméret generálunk. Ebben for ciklusokkal haladunk az előbb elmített tömbben, ezzel vizsgáljuk hol van a labda helye. A ciklus futásával halad a labda, ami ha eléri valamelyik tengely falát, akkor az adott iránnyal ellentétes irányba kezdi el a mozgást. Ekkor a folyamat előjele is változik. Ez még formalizálásra kerül.

## 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 forrása:

Szóhossz feladat megoldása.

Ebben a feladatban meg kellett vizsgálnunk az int típusnak az értékét a gépünkön. Ebben a programban két változóval kell dolgoznunk. Az egyik int értéke nulla, a másik pedig a 0x01, ami a következő sort jelöli. Ezután egy do while ciklussal növeljük a h értékét folyamatosan, egészen addig bitshift/bitwise-olnunk eggyel, amíg el nem jutunk az n által jelölt értékig. Ekkor a program megáll, és megkapjuk a bit értékét. Ebben az esetben balra bitwise-olásról van szó.

## 2.6. Helló, Google!

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

Megoldás forrása:

PageRank feladat.

Ez az algoritmus a Google nevezetű cég által lett kifejlesztve, melynek céja az oldalak "jó"-ságát mutatja be, ezalatt értve a minőségét, amit az adott oldalra mutató linkek száma határozza meg. Ezáltal a ranglistán egyre előrébb kerülve, a keresőben is elsőbbséget élvezhet az adott weboldal. Az emögött rejlő elméletet értem, a kódját viszont még nem teljesen, ezért leírást róla adni még nem szeretnék, hogy mikor majd megértem, akkor jól struktúrálva, és értelmesen megfogalmazva tudjak róla leírást adni. A későbbiekben teljesen ki lesz dolgozva a feladat.

## 2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás forrása:

Ikerprím feladat.

Maga a tétel az ikerprímekről szól, amelyek definiálva úgy tekinthetőek, mint az egymást követő prímek amik között a különbség kettő. A feladat megoldása során deklaráljuk a prímeket egy adott számig, majd a

differenciáljukat nézve, a nagyobból kivonjuk a kisebbet. Egy feltétellel a diferenciált keresünk a kettesre. Páronként deklaráljuk a kapott prímeket és a tételnek megfelelően reciprokkal szorozva adjuk össze őket egymással. Legvégül pedig a programon belül egy összesített értéket adunk vissza. Bezárólag pedig, matlab könyvtár által a megadott módon tudjuk ábrázolni a kapott eredményt.

## 2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás forrása:

A Monty Hall probléma megoldása.

Maga a Monty Hall probléma egy paradoxon, amelyben feltétel szerint van egy játékos, egy műsorvezető és a játékosnak három ajtó közül kell választania, amelyekből csak az egyik rejti a nyereményt. Miután a játékos választott egy ajtót, a műsorvezető kinyit egy olyan ajtót ami mögött nincs nyeremény, mivel hogy ő tudja melyik ajtó mit rejt. Ezek után megkérdezi a játékost, hogy szeretne-e a választásán módosítani, és ekkor jön a lényeg. A probléma azt tartalmazza, hogy a változtatással megnő a győzelem esélye 1/3-ról 2/3-ra. Ez azért van, mert azt feltételezzük, hogy a műsorvezető az általunk nem választott két ajtó közül, azért azt nyitotta ki, amelyiket, mert a másik mögött van a jutalom. A programunk úgy épül fel, hogy először is meghatározzuk a kísérletek számát, ami bármennyi lehet, majd létrehozzuk a kísérletek és a játékosok nevű változót, ami tartalmazza a választások számát egytől háromig, és a replace=T-vel pedig engedélyezzük, hogy a tippek száma többször is előfordulhasson a kísérletek során. A műsorvezető pedig a kísérletek számával lesz megadva. Egy for ciklussal megyünk végig a kísérleteken, ebbe viszont implementálnunk kell egy if elágazásokkal vizsgáljuk ki a választásokal. Az első if-ben a nyertes lehetőség értéke megegyezik a játékos tippjével, ekkor a műsorvezető a három lehetőségből kivonja a kísérlet értékét.Más esetben pedig, amikor nem találja el egyből a győztes ajtót, akkor a három lehetőségből kivonja a tippet és ami mögött a nyeremény van. Végül pedig kiírja az így kapott értéket. Következőnek pedig meghatározzuk, hogy változtat-e a játékos vagy nem. A nem esetén megegyezik a kísérlet a játékos tippjével. Ha pedig változtat, akkor egy for ciklussal kiválasztjuk azt, ami nem az eredeti választás, és nem is a műsorvezető által nyitott ajtó. A program pedig a kísérletek számának kiíratásával, a nem vátoztatással való nyereség és a változtatással való nyereség hányadosát, és a kísérletek számához való összes nyereség számát.

# 3. fejezet

# Helló, Chomsky!

## 3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása:

Ábra a feladathoz. Unárisba váltó Turing gép

A feladat lényege, hogy egy olyan programot hozzunk létre, ami a tízes számrendszerből, azaz decimálisból váltsunk át egyes számrendszerre, azaz unárisba pozitív egész számokat. Először is meghívjuk az alap könvtárakat, majd deklarálunk két integer változót. Bekérünk egy egész számot, amit egy for ciklussal alakítunk át. Ami megy sorra, és húzza a vonalakat, majd ha a szám öttel osztható lesz maradék nélkül, akkor egy szóközt helyez le.

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

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

Megoldás videó:

Megoldás forrása:

(A feladat megoldása során a Dicső mentorom, Pataki Donát volt)

Első: S̈ (S → aXbc) aXbc (Xb → bX) abXc (Xc → Ybcc) abYbcc (bY → Yb) aYbbcc (aY → aa) aabbcc

Második: S (S → aXbc) aXbc (Xb → bX) abXc (Xc → Ybcc) abYbcc (bY → Yb) aYbbcc (aY -> aaX) aaXbbcc (Xb → bX) aabXbcc (Xb → bX) aabbXcc (Xc → Ybcc) aabbYbccc (bY → Yb) aabYbbccc (bY → Yb) aaYbbbccc (aY → aa) aaabbbccc

## 3.3. Hivatkozási nyelv

A [KERNIGHANRITCHIE] könyv C referencia-kézikönyv/Utasítások melléklete alapján definiáld BNF-ben a C utasítás fogalmát! Majd mutass be olyan kódcsipeteket, amelyek adott szabvánnyal nem fordulnak (például C89), mással (például C99) igen.

Megoldás forrása:

Nyelvi szabvány

(A feladat megoldásában Pankotai Kristóf.) A program a két nyelvi keret változását mutatja be. A C99 szabvány egyik legjelentősebb újítása az volt, hogy a felhasználó immáron képes a C++ nyelv típusú kommentelését használni.

## 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:

Saját lexikális elemző

Ez a program a lefuttatása során egy másik programot állít elő. A lefuttatás menete: lex -o program.c program.l . A C-s programuk fordításánál pedig szükség van a végére illeszteni egy -lfl tag-et. A működési elve az, hogy az L nyelv lexelve létrehozza azt a C programot, amely egy komplex karaktersorból kitudja szűrni a valós számokat. A programot a két dupla % (százalékjel) szedi három részre. Az első harmada tartalmazza a C programba kerülő részt. A középső része tartalmazza a szabályrendszert, és a C-s ciklust. A végső harmadban pedig a komplett main rész van implementálva.

## 3.5. Leetspeak

Lexelj össze egy l33t ciphert!

Megoldás forrása:

Saját l33t chiper

A programunk az L felépítés során, egy másik C forrást hoz létre, melyet a -lfl parancssal kell lefordítani. A kódsor lényege, hogy a betű- és számkészletünkhoz rendel hasonló karaktereket, melyet a futás során kicserél. Ezt úgy éri el, hogy minden betűhöz és számhoz hozzárendel egy négy tagú mátrixot, amelyben jelöljük az általunk vélt megfelelő hasonmás karaktereket.

## 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);
```

i. -> Azt jelenti, ha eddig nem volt figyelmen kívül helyezve a SIGINT jel, akkor a jelkezelő függvény kezelje. Fordított esetben hagyjuk figyelmen kívül.

ii. -> Adott egy for ciklus, amelyben az i értékét növeljük egyesével, amíg az értéke eléri a négyet.

iii. -> Ez megegyezik az előzővel, ugyanis for ciklusban nem lesz más értéke, csak a feldolgozás ideje különbözik a két megadott növelési formánál.

iv. -> Ez egy hibás program, mert egy időben deklaráljuk az i-t és hivatkozunk rá tömbként. A végrehajtás sorrendjében van a hiba, mert a sorrendiség nem helyes, nem ismerjük.

v. ->

vi. ->

vii. ->

viii. ->

A feladat teljes megoldása még kidolgozás alatt, mert nem teljesen értem a további függvényeket.

## 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

Matlag

A feladat megértéséhez először is szükségünk van az elsőrendű logika ismeretére, ugyanis az adott példa alapján, ez négy logikai állításból áll. A legfontosabb megérteni az alap kifejezéseket. A foreall jelzi az univerzális kvantort, azaz a bármely-bármelyik, az exist, a létezik kifejezést jelöli. A wedge az implikáció és a supset pedig a konjukció és a neg pedig a negált.. A program pedig AR nyelven íródott.

I. -> Bármely x esetén létezik olyan y, amelynél ha x kisebb, akkor y prím szám lesz.

II. -> Bármely x esetén létezik olyan y, amelynél ha x kisebb, akkor y prím szám lesz, és ha y prím szám, akkor annak második utánai is prím szám lesz.

III. -> Létezik olyan y, amelynél bármely x esetén az x prím ,és x kisebb, mint y.

IV. -> Létezik olyan y, amelynél bármely y kisebb x esetén az x nagyobb, és x nem prím.

## 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 forrása:

A feladatok deklarálva

A feladat megoldásában más segített, tutorált, de még így sem értem igazán, szóval majd befejezem.

# 4. fejezet

# Helló, Caesar!

## 4.1. double ** háromszögmátrix

Írj egy olyan `malloc` és `free` párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárban!

Megoldás forrása:

Mátrix megoldása

A programunk egy négyzetes mátrixról szó, amelynek a fő tulajdonsága az, hogy oszlopainak és sorainak száma megegyezik, és a főátlója alatt csupa nullákat kapunk, bár a kódunkban majd az átló alatti rész lesz csupa nulla. Először is deklaráljuk a két fő válrozónkat, ami a sorok száma lesz, és a tm mutatót, ami double típusú. Az if-ben szereplő részt megírva, a tm változó eredménye a malloc paranccsal megkapott tárterületet, ami a sor, szorozva 8 bájtnyi mérettel rendelkezik. A malloc-ot rá lehet kényszeríteni, hogy bármit visszaadjon, mert alapesetben void*-ot adna vissza. Ezután vizsgáljuk, hogy a malloc tud-e egyáltalán területet foglalni, mert ha nem, akkor a return -1-e visszaugrunk a program elejére. További kidolgozás alatt.

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása:

C Exor feladat.

C Exor feladat github-on.

Ez a feladat a titkosításról szól, melynek a kulcsa az EXOR, azaz a kizáró vagy logikai elem. A program működéséhez szükségünk van, egy bárki által elkészített szöveget tartalmozó fájlra, mondjuk egy txt-re, amit majd a program átkódol az általunk meghatározott kulccsal, ami egy nyolc karakteres sztringként lesz kezelve, ami azért fontos, mert a vissza kódoláshoz is ekkora hosszúságú kulcs kell, amivel majd a későbbiekben dekódolni is tudjuk. Maga a forrásnak a megírását a kulcs méretének és a buffer méretének meghatározásával kezdjük, amelyek konstansok lesznek. A main-be deklarálnunk kell a kulcs és a buffer tömböket, amik a szükséges kulcsot és az eredményt tárolják. A deklarált két int, a a kulcsunk aktuális

részét/elemét mutatja, amivel végbe megy majd a művelet, és a második pedig, a beolvasott bájtoknak az összegét mondja meg. A harmadik int, azaz a kulcs_meret során használjuk strlen és a strncpy függvényekre, amelyeknek a lényege, hogy lerögzíti a hosszát stringben, amíg a ncpy pedig a végső másolatát rögzíti szintén stringben. Ezután egy végső while és for ciklussal folyamatosan olvasunk a bemenetről, és tároljuk a beolvasottakat a bufferben, amíg már nem tudunk több mindent beolvasni. Ekkor a ciklusban az olvasó 0 értéket ad vissza, amivel a program véget ér. A program megírása után a gcc-vel lefordítom, majd a futtatáshoz a ./programneve kulcs <a szöveges fájl, amiből kódolunk> "a fájl, amibe a titkosított szövegrészlet kerül bele".

## 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása:

Ezen az oldalon található meg a megoldás.

Ebben az esetben ugyan azt akarjuk elérni, mint a C nyelvű titkosító helyzetében. Azaz hogy az általunk, vagy a felhasználó által meghatározott kulccsal egy szintén saját választott szöveges dokumentumot kódolunk le. A szerkezete és működése ezáltal azonos lesz, az előbb említett nyelvben megírt társával. Fordítani a javac programneve.java . Majd a program futtatásához a köverkező kell: java programneve kulcs > titkosított.szöveg .

## 4.4. C EXOR törő

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

Megoldás forrása:

C Exor törés.

C Exor törés github-on.

Ez a program, az előbb megcsinált kódoló programunk által legenerált szöveges fájlt tudja dekódolni az általunk előzőleg megadott kulccsal, amely nyolc karakterből álló szting. A program alapvető definiálásból indul, ami a maximális szöveg terjedelmet, a bájtokban mért memória tárolást, a kulcs méretét, ami megint csak az említett nyolc karakterből álló sztring lehet, és végűl a szabadszoftverű operációs rendszerforrást. Ezek után az első változónk, az átlagos szóhossz, amit a beleimplementált for ciklussal a szóközök segítségével számítunk ki. Ebben az esetben az "sz" változó, a szóközök számát jelenti. A számítás végén a return utasítással úgy adjuk vissza az értéket, hogy elosztjuk a hosszúságot a szóközök megszámlált értékével. A következő szegmens a szöveg tisztaságának vizsgálata, amivel csökkentjük a törések potenciálját. A középső komponens maga az exor része lesz, amiben a for ciklussal bájtonként hajtkuk végre a műveletet. A benne levő "%" jel által lesz a kulcs mindig aktuális. Az elkövetkezendő részlet pedig a törés végrehajtása lesz. Ehhez be kell olvasni a titkos fájlt, amit egy while ciklussal érünk el addig, amíg csak van adat, ha már nincs, akkor a read 0 értékkel tér vissza, és leáll az utasítás. A rengeteg egybeágyazott for ciklussal elérjük, hogy az összes lehetséges kulcs álljon elő. Végül, ha sikerül az exortörés, kiírjuk a kulccsal a tiszta szöveget. A lefordítása alapesetű, és a futtatása megegyezik a titkosító programéval.

## 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

Még nem látom át igazán a programot, de kidolgozás alatt áll.

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó: https://youtu.be/XpBnR31BRJY

Megoldás forrása:

Visszaterjesztés.

(A feladat megoldásában Dicső tutorom, Pataki Donát segített)

A perceptron nem más mint egy algoritmus model, ami az emberi agy muködését próbálja utánozni. Hasonló a neurális hálóhoz azonban van pár különbség. Ugyanúgy input után elkezd varázsolni és jobb esetben és megfelelo mintavétel után helyes eredményt ad vissza. Azonban a közbelső értékeknek van súlya amit ḿég adott konstansokkal is ki lehet egészíteni. Az így kapott súlyokat összeadja és ha ez elér egy bizonyos szintet, akkor a program adott része aktiválódik. És egy a lineáris folyamat ismétlodik amíg el nem jut a v́álaszig.

# 5. fejezet

# Helló, Mandelbrot!

## 5.1. A Mandelbrot halmaz

Írj olyan C++ programot, amely kiszámolja a Mandelbrot halmazt!

Megoldás forrása:

Mandelbrot halmaz kiszámolása.

Mielőtt a feladatban szereplő halmazzal dolgoznánk, tisztáznunk kell, hogy mi az a fraktál, és milyen kapcsolatban állnak a Mandelbrot-halmazzal. A fraktálok lényegében olyan alakzatok, melyek végtelenül komplexek. Két fő tulajdonságuk van, az egyik, hogy a legtöbb geometria alakzattal ellentétben a fraktálok szélei "szakadozottak", nem egyenletesek. A másik tulajdonságuk pedig, hogy nagyon hasonlítanak egymásra. Ha egy kör határfelületét folyamatosan nagyítjuk, egy idő után kisimul(a csúcsokat leszámítva), megkülönböztethetetlenné válik egy egyenestől. Ezzel szemben a fraktálok első tulajdonsága, mi szerint határfelöletük szakadozott, megmarad, függetlenül a nagyítás mértékétől. A Mandelbrot halmaz is a fraktálok közé tartozik. Ezt és a hozzá tartozó szabályt Benoit Mandelbrot fedezte fel 1979-ben. A halmaz komplex számokból áll, és az ezekből álló sorozat konvergens, azaz korlátos. Ezeket a számokat ábrázolva a komplex számsíkon kapjuk meg a Mandelbrot-halmaz híres farktálját.

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

Írj olyan C++ programot, amely kiszámolja a Mandelbrot halmazt!

Megoldás forrása:

Mandelbrot halmaz std::complex osztállyal.

Az előző feladatot fogjuk megoldani, csak most egy kicsit másképpen. Ahogy láttad, az előbb a komplex számokat két változóban tároltuk, egyikben a valós, ,másikban pedig a képzetes részét. De az infromatikusok lusták, mindek használnánk 2 változót, ha lehet egyet is. Ezt teszi számunkra lehetővе a `complex` library, melynek segítségével a gép képes kezelni ezeket a számokat. Na itt néhány dolog eltér az előző feladathoz képest. Itt használjuk elsőnek a complex típust, ami `double`-ket tartalmaz, és két részből áll, a valós és az imaginárius részből. Ennek a segítségével definiáljuk a c és a z_n változókat. Majd, innen már ismerős lehet, kiszámoljuk minden c esetén a z_n-eket, és ha elérjük az iterációs határt akkor, tudhatjuk, hogy az iteráció konvergens. Ebből következik, hogy a c eleme a Mandelbrot halmaznak. A while

fejrészében látható `abs()` függvény az abszolút értékét adja meg az bemenetként kapott argumentumának. A halmazt lértehozó sorozat képzési szabálya egy az egyben beírható a programba, nincs szükség semmilyen szétbontásra, mint az előző programnál volt, köszönhetően annak, hogy képesek vagyunk kezelni a komplex számokat. Pusz dolog, hogy a a program a futása azt is látjuk, hogy hány százalékát végezte el a számításoknak a gép. Végezetül pedig itt is kiírjuk a png fájlt a parancssori argumnetumként megadott fájlba a `write` segítségével.

## 5.3. Biomorfok

Megoldás forrása:

Biomorfok

A biomorfokra Clifford Pickover talált rá, aki a természet egyik törvényének a feldedezésének hitte ezt a jelenséget. Ezt Julia halmaznak is nevezzük. A két halmaz között a különbség, hogy a Mandelbrot során a c egy változó, amíg a biomorfok esetén egy állandót alkot majd, azaz a z vizsgálatai során, ugyan az marad.

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: `bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu` nevű állománya.

## 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:

Mandelbrot nagyító és utazó C++ nyelven.

Telepíteni: sudo apt-get install libqt4-dev

A program a QT GUI-t használja, ennek segítségével tudjuk elkészíteni a Mandelbrot halmazt beutazó programunkat. Ez a GUI az egyik legertejedtebb garfikus interfésze a C++-nak, rengeteg tutorial van róla fent a neten.

Fordítás: Az szükséges 4 fájlnak egy mappában kell lennie. A mappában futtatni kell a `qmake -project` parancsot. Ez létre fog hozni egy `*.pro` fájlt. Ebbe a fájlba be kell írni a következőt: `QT += widgets` sort. Ezután futtatni kell a `qmake *.pro`. Ezután lesz a mappában egy `Makefile`, ezt kell majd használni. Ki adjuk a `make` parancsot, mely létrehoz egy bináris fájlt. Ezt pedig a szokásos módon futtatjuk. Rengeteg figyelmeztetést ad vissza, de ezzel most nem kell törődni, hiszan a bináris fájl elkészült, melyet futtatunk, és elindul az utazásunk a végtelenbe. Ahhoz, hogy részletesebb képet kapj a ránagyított területről, az "n" billentyűt kell lenyomnod, mely kiszámolja a z-ket a megadott területen. Itt lehet látni, hogyan mosósdik össze a Mandelbrot és a Biomorfos téma. A hatmadik kép, már majdnem olyan, mint egy biomorf.

## 5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: https://youtu.be/Ui3B6IJnssY, 4:27-től. Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#id570518

# 6. fejezet

# Helló, Welch!

## 6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... térj ki arra is, hogy a JDK forrásaiban a Sun programozói pont úgy csinálták meg ahogyan te is, azaz az OO nemhogy nem nehéz, hanem éppen természetes neked!

## 6.2. LZW

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

Megoldás videó:

Megoldás forrása:

## 6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

## 6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

## 6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

## 6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

# 7. fejezet

# Helló, Conway!

## 7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

# 8. fejezet

# Helló, Schwarzenegger!

## 8.1. Szoftmax Py MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.5. Minecraft-MALMÖ

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

# 9. fejezet

# Helló, Chaitin!

## 9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

## 9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

## 9.3. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

## 9.4. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

## 9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 9.6. Omega

Megoldás videó:

Megoldás forrása:

# 10. fejezet

# Helló, Gutenberg!

## 10.1. Programozási alapfogalmak

[?]

1.2. A magas szintű nyelven megírt programokat forrásszövegnek, a nyelvtani szabályokat pedig szintaktikai szabályoknak nevezzük. Frodítóprogrammal tudjuk fordítani, lexikális egységekre bontja tárgyprogramot hoz létre. Ezt majd futtatni kell, amit a futtató rendszer felügyel. Az interpretes technika nem készí tárgyprogramot, utasításonként elemz a programot, értelmezi és végrehajtja. Minden nyeévnek megvan a saját szabvány, a hivatkozási nyelve. Az implementációk a forsítóprogramok vagy az interpreterek, egy platformon több is lehet belőlük.

1.3. Imperatív nyelvek: algoritmus mely mülködteti a processzort, utasítások, változók, eljárásorientált- és objektumorientált nyelvek. Deklaratív nyelvek: nem kötődnek a Neumann architektúrához, funkcionális- és logikai nylevek

1.4. Jelölésrendszerek: terminális, nem terminális, alternatíva, opció, iteráció. Ezekkel szintaktikai szabályoka lehet írni: bal oldalon nem formális, :, majd elemsorozat. Programozni megtanulni papíron nem lehet.

2.4. Az adattípus absztrakt programozási eszköz, más eszköz komponenseként jelenik meg. Az adattípust a tartomány (amilyen típust felvehet), a művelet (amit a tartomány elemein elvégezhet) és a reprezentáció (az értékek tárban való megjelenése) határoz meg. A programozó is definiálhat típusokat. Az alaptípusból tudunk más típust (alaptípust) származtatni, szűkített tartománnyal de ugyanolyan műveletekkel.

2.4.1. Az egész típus belső ábrázolása fixpontos, a valósé lebegőpontos. A két típus együtt a numerikus típusok, mivel numerikus műveletek hajthatók rajtuk végre A karakrer típus elemei a karakterek, a karakterlánccé pedig a sztringek. A logikai típus tartományának elemei a hamis és az igaz érékek. A felsorolás típust saját típusként lehet létrehozni, megadva atrtomány elemeit. Sorszámozott típusnak számít az egesz, karakteres, logikai és felsorolás típusok (minden elemnek van megelőzője és rákövetkezője). A sorszámozottak egy alaptípusa az intervallum típus.

2.4.2. Összetett típus a tömb és a rekord. A tömb elemei azonos típusuak, lehetnek egy vagy többdimenziósak. A tömb elemeire indexek tségével tudunk hivatkozni. Absztrakt adatszerkezetek folytonos ábrázolására alkalmas. A rekord tartományának elemei értékcsoportok, melyek elemei (mezői) különböző típusuak lehetnek. Minden mezőnek van neve és típusa. A C és a Pascal esetében is a rekord típusa egyszintű (nincsenek almezők), de a mezők típusa lehet öszetett. A rekord nevével (eszköznév) az összes mezóre tudunk hivatkozni, ha csak egyre akarunk akkor: eszköznév.mezőnév.

2.4.3. A mutató típus tartományának elemei tárcímek, így valósítjuk meg az indirekt címezést (minden mutató egy címre mutat). Van egy speciális eleme, a NULL, ami nem mutat sehova.

2.5. A nevesített konstansnak van neve, típusa és értéke. Mindig deklarálni kell. Szerepe, hogy a sokszor előforduló értékeket beszélő névvel látjuk el, így könnyebben tudunk utalni rá a szövegben, illetve ha az értékét ki akarjuk cserélni, akkor ezt elég egy helyen megtenni.

2.6. A változónak van neve, attribútuma, címe és értéke. A kódban mindig a név jelenik meg, ehhez kölönböző módon rendeljük hozzá a másik hármat. Az attribútum a változó viselkedését határozza meg a futás alatt (ez többnyire a típust jelenti). Háromfajta deklarációt különböztetünk meg itt: explicit, implicit és autómatikus. A változó címe a tárnak azt arészét határozza meg, ahol a vátozó értéke van. A cím rendelhető: statikus, dinamikus és a programozó által vezérelt tárkiosztással. A változó értéke a címen elhelyezkedő bitkombináció. Az érték meghatározható: érékadó uatsítással. A kezdőértékadás lehet: explicit (a programozó választ a típusnak megfelelő tetszóleges értéket) és autómatikus (a programtól kap 0-áskezdőértéket).

## 10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

1.1. Annak a módja, hogy egy programozási nyelvet elsajátítsunk, nem más mint higy programokat írjunk az adott nyelven. A C nyleg tanulását is kezdjük úgy, mint más nyelvek tanulása esetén, a legegyszerűbb programmal a Hello World magyar változatával. Fordítjuk: cc figyel.c, majd futtatjuk: ./a.out. Mivel a cc-vel fordítottuk ezért egy alapértelmezett kimenetbe megy az a.out-ba. A C programok tetszőleges nevű függvényeket tartalmaznak amelyek a számítási műveleteket határozzák meg. Speciális függvény a main (főprogram) mely minden programban elő kell forduljon. Itt hívjuk meg az előre megírt függvényeket, vagy itt írjuk meg. A program olyan sorrendben végzi el az utasításokat ahogy azok a main-be vannak. A függvény neve után szereplő () az argumentum listát tartalmazza, a {} pedig az utasítás listát (lehetnek üresek is). A printf("Figyelem emberek\n") függvényhívás a kimenetbe írja a Figyelem emberek szöveget, a végére pedig egy sortörést.

1.2. Egysoros kommentet a //-el, többsorosat pedig a /* */ jelek segítségével hozunk létre. Az int az egész, a float a lebegő, char a karakter, stb típusú váltózókat jelöli. Sorok végét ;-vel zárjuk le. A while ciklus működése: akkor áll le, amikor a feltétel hamissá válik. A printf-ben lévő %-al meg tudjuk adni, hogy a kiírandó szám milyen típusú lesz.

1.3. A for ciklus sokban hasonlít a while-hoz, ő is akkor lép ki a ciklusból, ha a feltétel hamissá válik.

1.4. A #define segítségével a program elején szimbolikus állandót vagy változót adunk egy karakterlánchoz.

1.5. A getchar() függvény híváskor karaktert olvas be, a putchar() hívásakor pedig karaktert ír a kimenetre.

## 10.3. Programozás

[BMECPP]

A könyv első részében (16.oldalig) a C++ nem objektumorientált különbségei, újításai vannak a C nyelvhez képest.

Függvényparaméterek és visszatérési érték: az üres paraméterlista azt jelenti, nincs a függvénynek paramétere, nem pedig azt, hogy tetszőleges számú paraméterrel hívható.

A main függvénynek tudunk parancssor argumentumokat adni, illetve nem kötelező a return 0; használata, ez alapértelmezés.

A bool típus használata átláthatóbb, praktikusabb kódot eredményez.

A C++-ban mindenhol állhat deklaráció, ahol utasítás is.

Függvények túlterhelése: ugyanolyan néven több függvényt is létrehozhatunk, amennyiben különbözik az argumentumlistájuk. A függvényt nem csak a neve, hanem argumentumlistája is meghatározza, ezáltal a névadás egyszerűbbé válik.

Alapértelmezett függvényargumentumok: megadhatunk alapértelmezett függvényargumentumokat is, melyek híváskor elhagyhatók. Arra viszont ügyelnünk kell, hogy milyen sorrendben hagyhatók el.

Refetípussal történő paraméterátadás: a cím szerinti átadás egy modszere, mely sokat egyszerűsíthet a függvényünkön, nem kell mindenhol mutatót használni, ha referenciaátadást alkalmazunk. Itt is vigyázni kell néhány dologra, pédául arra, hogy egy függvényben sose adjunk vissza pointert vagy referenciát lokális változóra vagy érték szerint átadott paraméterre.

# III. rész

# Második felvonás

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

# 11. fejezet

# Helló, Arroway!

## 11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

# IV. rész

# Irodalomjegyzék

## 11.3. Általános

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

## 11.4. C

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

## 11.5. C++

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

## 11.6. Lisp

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

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

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