# Univerzális programozás

## Írd meg a saját programozás tankönyvedet!



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

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>Univerzális programozás | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Bátfai, Norbert Ã©s Fazekas, Márk | 2019. április 29. | |

| 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

# Előszó

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

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

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

## Hogyan forgasd

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

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

### Milyen nyelven nyomjuk?

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

### Hogyan nyomjuk?

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

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

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

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

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

# I. rész

# Bevezetés

# 1. fejezet

# Vízió

## 1.1. Mi a programozás?

## 1.2. Milyen doksikat olvassak el?

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

## 1.3. Milyen filmeket nézzek meg?

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

# II. rész

# Tematikus feladatok

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

# 2. fejezet

# Helló, Turing!

## 2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása: https://gitlab.com/nagyfazek06/programozas/blob/master/vegtelena.c https://gitlab.com/nagyfa 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 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.

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 írj egy olyan programot, ami egy labdát pattogtat a karakteres konzolon! (Hogy mit értek pattogtatás alatt, alább láthatod a videókon.)

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

Megoldás forrása: https://gitlab.com/nagyfazek06/programozas/blob/master/ifbunce.c https://gitlab.com/nagyfaze

Tanulságok, tapasztalatok, magyarázat...

## 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Írj egy programot, ami megnézi, hogy hány bites a szó a gépeden, azaz mekkora az int mérete. Használd ugyanazt a while ciklus fejet, amit Linus Torvalds a BogoMIPS rutinjában!

Megoldás videó:

Megoldás forrása: https://gitlab.com/nagyfazek06/programozas/blob/master/szohossz.c

Egy egszerű bitshifteléssel végig megyünk egy int-en aminek a mérete álltalában 32.

## 2.6. Helló, Google!

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

Megoldás videó:

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

A Pagerank egy olyan algoritmus amely az oldalak fontosságát lehet megállapítani. Az algoritmus megvizsgálja az oldalakat majd eldönti melyik a leggyakrabban látogatott oldal. Böngészés során ennek segítségével találhajuk meg a bizonyos 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 átékvezető (aki tudja hol mi van) kinyit egy ajtót (mindig olyat nyit ami mögött kecske van). Az ajtó kinyitása után a játékvezető megadja a lehetőséget, hogy a játékos meggondolja a választását. Ekkor dönti el a játékos hogy vált e vagy sem és a döntés után a játékvezető kinyitja a választott ajtót, s a játékos megkapja a nyereményét. A paradoxon kérdése, hogy megéri-e váltani.

# 3. fejezet

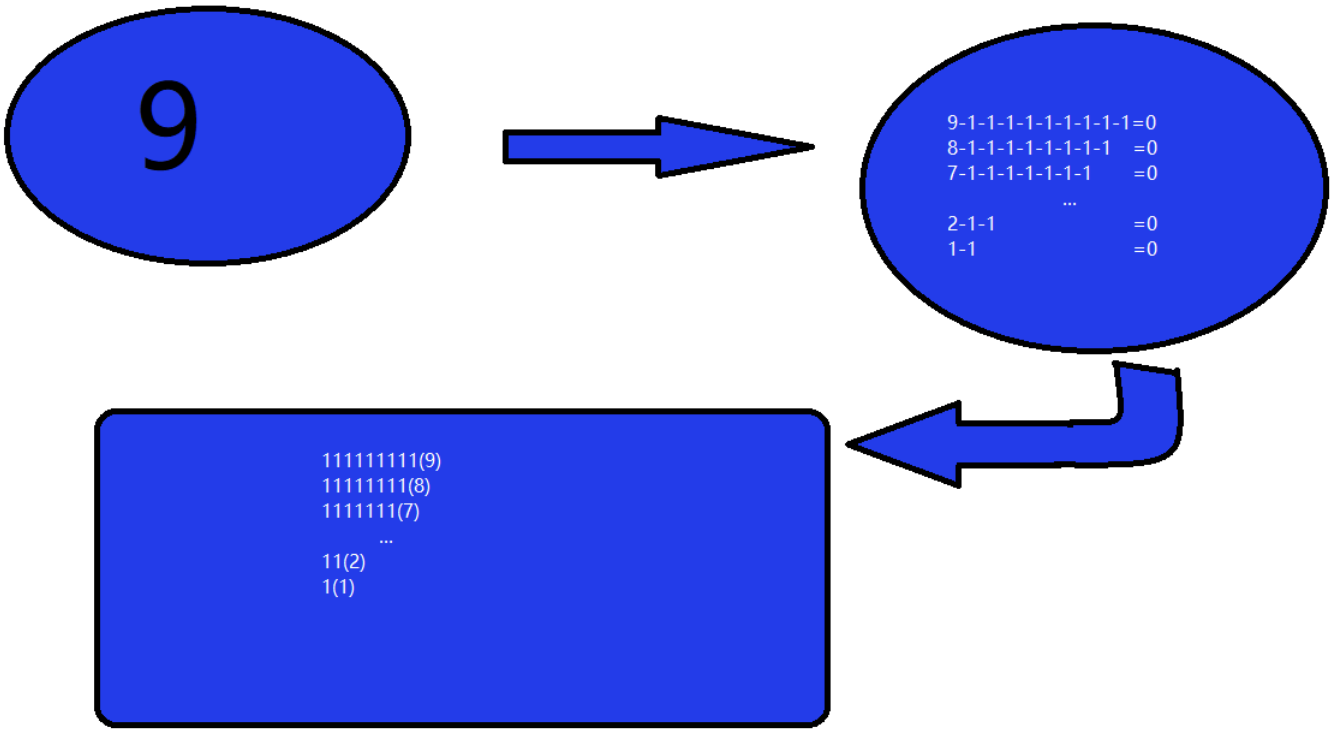
# Helló, Chomsky!

## 3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás




Tanulságok, tapasztalatok, magyarázat.. Az unáris számrendszerben való ábrázolás n darab egyforma jel, karakter egymás utáni leírásával történik. A decimálisból unárisba átváltás: folyamatosan 1-eket vonunk ki a számból 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 -> aBSc
            S -> abc
            Ba -> aB
            Bb -> bb
                Levezetés: S -> aBSc -> aBaBabccc -> aaBBabccc -> aaBaBbccc ↩
                    ->  aaaBBbccc -> aaBbbcc -> 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 -> aabbYbccc -> aabYbbccc  -> aaYbbbccc ->  ↩
                   aaabbbccc
```

Tanulságok, tapasztalatok, magyarázat... Noam Chomsky alkotta meg a generatív nyelvtan elméletét és ő dolgozta ki a Chomsky-hierarchiát. A gramatikának 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

Tanulságok, tapasztalatok, magyarázat... 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 szab- vánnyal, 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 kapcsolot 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 haszálni az stdoutra 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 eseté ben, 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 írt, 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.l

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

## 3.6. A források olvasása

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

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

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

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

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

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

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

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

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

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

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

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

Megoldás forrása: https://gitlab.com/nagyfazek06/programozas/blob/master/jelkezelo.c

Megoldás videó:

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

## 3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

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 prímszámra 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 programban helyet foglalok a memóriában egy double-öket tartalmazó alsó háromszögmátrixnak. Majd k=0-tól a mátrix minden elemének 1.1-es lépésközzel értékül adtam k-t. Ezután a mátrix standard uotputra való kiíratása történik. Következőnek felszabadítom a pointerek által lefoglalt memóriacímeket, majd a "return 0"-val jelzem az operációs rendszer felé, hogy a program futása hiba nélkül befejeződött.

## 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 végzett 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, iletve 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. Ezalól kivétel az utolsó, amiről lentebb szó lesz.

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: https://gitlab.com/nagyfazek06/programozas/blob/master/perceptor.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 progvam 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 néhány változót a precompiler számára, amelyek a kimeneti kép méretét határozzák meg, illetve vizsgált tartományt a komplex számsíkon. A GeneratePNG függvény paraméterként megkapja a program által generált kép adatait pixelenként a tomb nevű int típusú N*M-es mátrixban tárolva. A függvényen belül létrehozunk egy N*M pixeles PNG kiterjesztésű képfájlt, amelybe az egymásba ágyazott for ciklus pixelenként beletölti az adott pixelre vonatkozó adatokat, majd a for ciklus után kiírja a képfájlt lemezre "kimenet.png" néven. A main függvényen belül létrehozzuk az N*M-es mátrixot, amelyben tároljuk a kép pixelenkénti adatait, beállítjuk a komplex számsíkon való lépegetés lépésközét a "dx" és "dy" változóban, majd létrehozunk három Komplex típusú változót, amely a komplex számokat fogja tárolni a Mandelbrot-halmaz kiszámításához. Ezután belép a porgram a for ciklusba, ahol lépked a komplex számokkal a megadott tartományban a megadott lépésközzel, majd a benne lévő while ciklus meghatározza a kép megfelelő pixelének színét an- nak 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

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 akár élő organizmusok is lehetnének, viszont nem muszáj természetes eredetűnek lennie az alakzatnak (magyarán, akár lehetnek számítógép által generáltak is). 5.4 FT 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 iteréció 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 fel- használó 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 létrehozzuk a GUI-t, illetve megadjuk a paramétereit (méret, méretezhetőség), létrehozzuk a kontroll objektumokat, a gombokat, amelyekkel változtatható a vizsgált komplex szám tartomány. A `plotPoints` eljárás felelős a vizsgált tartomány bejárásáért és az alapján a halmaz elemeinek kiszámításáért. A `actionPerformed` eljárás felelős azért, hogy a felhasználói interakciót lehetővé tegye az által, hogy "megmondja", mi történjen, ha a felhasználó rákattint egy gombra.

# 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: c++ : https://gitlab.com/nagyfazek06/programozas/blob/master/osztaly.cpp java: https://gitlab.
java/tree/master

Létrehozzuk a polargen nevű osztályt ahol a konstruktorba megadjuk, hogy nincs eltárolt szám. A "kovetkezo" függvény megnézi, hogy van e tárolt szám. Ha nincs generálunk 2db számot. Az egyiket eltárolja majd a logikai változó hamis értéket vesz fel és a másik generált számmal tér vissza. Az objektum orientált programozás természetes, hisz a z 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 progra melején 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 az egyes 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 inorderben 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

Inorderben először a fa bal oldalát dolgozzuk fel, majd ennek végeztével a fa jobb oldalát.

## 6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: https://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 belűl 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 megoldósának 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 (esetünkben) 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, hogy a scope elhagyása után lefutó konstruktor ne törölje le az eredetileg létező fát.

# 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. 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` https://gitlab.com/nagyfazek06/programozas/blob/master/ant.java

A program elején importáljuk a szükséges csomagkat amelyek kezelik a az ablakot, egeret és billentyűzetet. A game_of_life egyetlen publikus osztályába található a main, a program belépési pontja. Létrehozunk egy RenderArea típusú változót, 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/eletgame

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.

# 8. fejezet

# Helló, Schwarzenegger!

## 8.1. Szoftmax Py MNIST

Python

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

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

A program elején importájuk a tensorflow modult, amely segítségével töltjük be az MNIST adatbázisban tárolt adatokat. Az x_train, y_train, x_test, y_test változókban eltároljuk az MNIST adatbázisban található képeket, és hozzá tartozó címkéket, illetve a tesztképeket és címkéket. Ezután konvertáljuk az x_train és x_test tömbök értékeit float típusúra, majd normalizáljuk az értékeket, hogy 0 és 1 közé essenek. Importálunk még hat modult, amelyek a neurális háló felépítéséhez szükségesek. Ezután létrehozzuk 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 importájuk a szükséges modulokat. Az importálások után létrehozunk négy függvényt, amelyek a programot hivatottak rövidíteni. A függvények után létrehozzuk a neurélis háló rétegeit. Ebben a programban 5 rétegű a neurális háló. A rétegek megadása után megadjuk, hogy milyen módon szeretnénk kiértékeltetni a bemenetet, majd megadjuk, hogy milyen algoritmussal tanítjuk fel a hálót. Ezután következik a tanítás, illetve minden századik iterációban leteszteljük a háló tudását.

## 8.3. Minecraft-MALMÖ

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

Megoldás forrása:

A Minecraft-MALMÖ projektet azért hozta létre a Microsoft, hogy a Minecraft játékon belül lehessen mesterséges intelligenciákat létrehozni, tanítani és tesztelni. Az én programomban alapvetően az a célja Steve-nek, a karakternek, hogy ne akadjon el 5 percen keresztül, miközben folyamatosan halad. Ezt úgy éri el többnyire, hogy megvizsgálja az előtte lévő 12 blokkot, és bizonyos feltételek teljesülése esetén ugrik, vagy elfordul és halad más irányban. Steve még közel sem tökéletes, ugyanis a lávatavakat és a szakadékokat még nem tudja kikerülni. A programom megírásához sokat felhasználtam a példaprogramokból. A programban először importálom a szükséges modulokat a program futásához, illetve a MALMÖ API-t. A `restart_minecraft` függvény arra szolgál, hogy a mission lefutása után a Minecraft tudja fogadni a következő missiont. A `run` függvényben található a mission pontos megadása, illetve a mission alatti vezérlés. a missionXML string-ben van megadva a világ generálásának szabályai, Steve kiindulási helyzete, illetve itt van megadva, hogy figyelje a körülötte lévő blokkokat. Ezután megadom, hogy 300 másodpercig fusson a mission, illetve, hogy 640x480-as felbontással fusson a Minecraft. Megadom, hogy Minecraft kliens milyen címen és milyen porton éri el, majd megpróbál a program csatlakozni. 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 át tudja vagy fel tudjon ugrani az akadályokra és ne folyamatosan ugráljon utána. A harmadik elágazás arra szolgál, hogy váltakozva jobbra és balra is forduljon Steve. A következő elégazás arra szolgál, hogy eldöntse, melyik égtáj felé néz Steve. Erre azért van szükség, hogy a megfelelő 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 az előtte lévő blokkokat, hogy mit csináljon (ugorjon vagy forduljon).

# 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

[?]

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

A programozási nyelvek három szintje van, a gépi nyelv ,ezek azok, amelyek már a processzor nyelvére vannak lefordítva, az assembly szintű nyelv, másnéven gépközeli nyelvek, illetve a magas szintű nyelvek, mint például a java és C++. Minde processzor saját gépi nyelvvel rendelkezik, ezért a forrás szöveget a porcesszor gépi kódjának megfelelő kóddá kell alakítani. Erre két megoldás létezik: a fordítóprogramos és az interpreteres. A fordítóprogramos megoldás a forrásszöveget lefordítja gépi kódra, majd ezután válik futtathatóvá, míg az interpreteres megoldás esetében az interpretes soronként halad végig a forráskódon és olvasási sorrendben hajtja végig az utasításokat, tehát itt nincs szükség futtatás előtti fordításra. Minden programozási nyelvnek van saját hivatkozási nyelve, azaz szabványa. Ebben vannak definiálva a szintaktikai és szemantikai szabályok, legtöbb esetben angolul. Léteznek implementációk, melyek operációs rendszereken való fordítóprogram-, vagy interpreter megvalósítást jelent. Ezek nem kompatibilisek egymással. Létezhet egy operációs rendszeren több implementáció is, ezek sem feltétlen kompatibilisek egymással. Manapság a programozáshoz IDE-ket használunk (Integrated Development Environment), amelyek grafikus programok, amelyekben általában van beépített szövegszerkesztő, fordító, futtatórendszer.

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

Az adatabsztrakció első formája az adattípus. Az adattípus rendelkezik névvel, amely azonosítja a típust, például int, double. Léteznek típusos és nem típusos programozási nyelvek. A típususosok engedik, hogy a programozó adja meg a változók típusát. Ilyenek például a C++ és a Java. A nem típusosok automatikusan állapítják meg a változó típusát. Ilyenek például a R és a Python. Adattípusoknak két csoportja van, az egyszerű és az összetett. Az egyszerű adattípusok azok, amelyeket nem lehet tovább bontani, például int. Az összetett típusok például a struktúrák vagy a felhasználó által definiált típusok.

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

A nevesített konstansok azt a célt szolgálják a programokban, hogy a konstansoknak olyan nevet adjunk, amely jelképezi annak típusát és értékét. Illetve másik célja, hogy sokszori használat esetén csak a definiálásnál kelljen változtatni az értékét, ha szükséges. Ezeket a konstansokat mindig definiálni kell.

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

A változónak négy komponense van: a név, az attribútomok, a cím és az érték. A név az egy azonosító, a másik három komponenst egy névhez rendeljük hozzá. A legfőbb atribútom, a típus, amely a változó által felvett értéket határolja be. A változóhoz az attribútumok deklarációk segítségével rendelődnek. A deklarációnak különböző fajtáit simerjük: Explicit deklaráció, Implicit deklaráció, Automatikus deklaráció. A változó címe meghatározza a változó értékének a helyét. A címrendelésnek három fajtáját ismerjük: a Statikus tárkiosztás, a Dinamikus tárkiosztás, és a programozó által vezérelt kiosztás. A változó értékének a meghatározására több opció is van: értékadó utasítás, kezdőérték adás.

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

C-ben az aritmetikai típusok az egyszerű típusok, a származtatottak az összetett típusok. A karakter típus elemeit belső kódok alkotják. Logikai típus nincs, a hamis az int 0 az igaz pedig az int 1. A struktúra egy fix szerkeztű rekord. A void tartománya üres. A felsorolásos típusok nem fedhetik egymást. Különböző elemekhez ugyanazt az értéket hozzárendelhetjük.

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

A kifejezések szintaktikai eszközök. A kifejezések formálisan három dologból állnak: operandusokból, operátorokból, kerek zárójelekből. Létezik egyoperandusú(unáris), kétoperandusú(bináris) és háromoperandusú(ternáris) operátor, ezek attól függnek, hogy egy operátor hány operandussal végzi a műveletet. A kifejezéseknek három alakja lehet: a prefix, az infix, a postfix. A folyamatot, amikor a kifejezés értéke és típusa meghatározódik, a kifejezés kiértékelésének nevezzük. A kifejezéseknek van két típusa: a típusegyenértékűség, és a típuskényszerítés. Azt a kifejezést, amelynek értéke fordítási időben eldől, és a kiértékelését a fordító végzi, azt konstans kifejezésnek hívjuk.

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

Az utasítások megalkotják a programok egységeit: az algoritmusok egyes lépései, a fordítóprogram ezzel generálja a tárgyporgramot. Két csoportjuk van: a deklarációs utasítások, és a végrehajtó utasítások. A deklarációs utasítások mögött nem áll tárgykód, a fordítóprogramnak szólnak. A végrehajtó utasításokból pedig a fordító generálja a kódot. A végrehajtó utasításokat csoportosíthatjuk: értékadó utasítás, üres utasítás, ugró utasítás, elágaztató utasítás, ciklusszervező utasítás, hívó utasítás, vezérlésátadó utasítás, I/O utasítás, egyéb utassítás. A vezérlési szerkezetet megvalósító utasítások: ugró utasítás, elágaztató utasítás, ciklusszervező utasítás, hívó utasítás, vezérlésátadó utasítás.

VII. heti előadás (78-84. oldal):

A paraméterátadásnak többféle módja is lehet, ezek nyelvfüggőek, hogy melyik nyelv melyiket alkalmazza.

Történhet érték szerint, mint a C-ben például. Ekkor a formális paraméter értékül kapja az aktuális paraméter értékét. Ennél a módszernél a függvényben nem lehet megváltoztatni a aktuális paraméter értékét. Lehet címszerinti a paraméterátadás. Ekkor a formális paraméter címe értékül kapja az aktuális paraméter címét. Ilyenkor a függvényben meg lehet változtatni az aktuális paraméter értékét. Lehet eredmény szerinti átadás is, ekkor a formális paraméter szintén megkapja az aktuális paraméter címét, de nem használja, csak a végén beletölti az adatokat. Létezik még érték-eredmény szerinti, ekkor másolódik a cím szintén, és használja is az adatokat, majd a függvény végén belemásolja a formális paraméterbe az adatokat.

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

Olyan adattípus, amely megvalósítja a bezárást vagy információ rejtést. Az ilyen típusú programozási eszeközök műveleteihez a specifikációi által meghatározott interfészen keresztül férhetük hozzá. Így az értékeket véletlenül vagy szándékosan nem ronthatjuk el(biztonságos programozás). Az elmúlt évtizedekben nagyon fontos fogalommá vált és befolyásolta a nyelvek fejlődését.

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

A generikus programozás az újrafelhasználhatóság, és így a procedurális absztrakció eszköze. Bármely programozási nyelvbe beépíthető. A generikus programozás lényege: Megadunk egy paraméterezhető forrásszöveg-mintát, ami majd fordítási időben lesz kezelve. A mintaszövegből paraméterek segítségével előállítható egy lefordítható konkrét szöveg. Az újrafelhasználás alatt azt értjük, hogy egy mintaszövegből tetszőleges számú konkrét szöveg generálható, a mintaszöveg típussal is paramétertezhető. A generikus formális paramétereinek száma mindig fix. A paraméterkiértékelésnél a kötés az alapértelmezett, de alkalmazható a név szerinti kötés is. A paraméterátadás változónál értékszerinti, típusnévnél pedig névszerinti.

## 10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

V.heti előadás (Vezérlési szerkezetek című fejezet):

Egy nyelv vezérlésátadó utasításai az egyes műveletek végrehajtási sorrendjét határozzák meg. A C nyelvben a pontosvesző 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 ennek az értéke igaz, akkor az első utasítást hajtja végre, ha a kifejezés értéke viszont nem igaz, és van "else" rész, akkor a második utasítás hajtódik végre. Általános szabály, hogy az "else" mindig a hozzá lehközelebb eső "if"-hez tartozik. A "switch utasítás is a többirányú programelágazás egyik eszköze. Összehasonlítja egy kifejezés értékét több egész értékű állandó kifejezés értékével, és az ennek megfelelő utasítást hajtja végre. A "switch" -ben sok "case" és egy "default" található. A "default" akkor hajtódik végre, ha egyik "case" ághoz tartozó feltétel sem teljesül. A "while - for" szerkezet először kiértékeli a kifejezést, ha ennek az értéke nem nulla, akkor az utasítás végrehajtódik, ez addig ismétlődik, amíg nulla nem lesz a kifejezés értéke. A "do - while" szerkezet először végrehajtja az utasítást,és csak utána értékeli ki a kifejezést. Ha a kifejezés értéke igaz, akkor az utasítást újból végrehajtják. Ez addig ismétlődik, amíg a kifejezés értéke hamis nem lesz. A "break" lehető teszi, hogy elhagyjuk a ckliusokat, még idő előtt(for,while, do, switch). A "continue" utasítás a "break" utasításhoz kapcsolódik. hatására azonnal megkezdődik a következő iteráció lépés. A "goto" utasítás, akkor előnyös, ha ki akarunk lépni egy több szinten egymásba ágyazott ciklusból(a "break" egyszerre csak egy ciklusból tud kilépni). A címke ugyanolyan szabályok szerint alakítható ki, mint a változók neve és mindig kettőspont zárja.

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

Az utasítások a leírásuk sorrendjében hajtódnak végre, általános a szintaktikai leírásuk, és számos csoportba sorolhatók: Címkézett utasítások, mint például a "case" és "default" címkéi a "switch" utasítással használhatók. A címke egy azonosító nélküli deklarált azonosítóból áll. Kifejezésutasítsok, 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 ehetséges 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 definiálunnk, akkor az tetszőleges számú paramáéterrel hívható. A C++-ban azonban az üres paraméterlista egy "void" paraméter megadásával ekvivalens. C nyelvben is létezik több bájtos sztring. C++-ban miinden 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 azaonosítja. Míg a C nyelv úgy hivatkozik egy függvényre a linker szintjén, hogy egy aláhúzást tesz a függvénynevek elé, addig a C++ az egyes fordítókra bízza a névferdítés implementálását. Cím szerinti paraméterátadás, ha a változó címét adjuk át, ebebn az esetben nem tudjuk megváltoztatni úgy a változót, hogy az értéke megmaradjon. Az érték szerinti paraméteradásnál viszont, készül másolat a változóról, így végezhetünk műveleteket úgy, hogy a változó értékét nem befolyásoljuk. A C++ referenciatípus bevezetése feleselgessé teszi a pointerek cím szerinti pareméterátadását.

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 zárás 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 destruktor célja, hogy lefusson, amikor az objektum megsemmisül, ezáltal akár felszabadítva a diamikus adattagokat. A dinamikus adattagok osztályon belüli pointerek, amelyeket futásidőben hozzuk létre dinamikus memóriafoglalással, ezért ezeket, ha már nincs rájuk szükség, de legkésőbb a destruktorban fel kell szabadítani. A friend osztályok, illetve függvények olyan osztályok, illetve függvénynek, 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 operátokról és azok túlterheléséről szól C++ nyelvben. Az operátorok alapértelmézés 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 erdeti 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.

# 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.