Univerzális programozás

Írd meg a saját programozás tankönyvedet!

DRAFT

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

PDF BY DBLATEX

Copyright © 2019 Dr. Bátfai Norbert

Copyright (C) 2019, Norbert Bátfai Ph.D., batfai.norbert@inf.unideb.hu, nbatfai@gmail.com,

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

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

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

http://gnu.hu/fdl.html

DRAFT

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Nyitrai, Dávid	2019. május 8.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.1.0	2019-02-22	Az első fejezet elkészült.	Nyitrai Dávid
0.2.0	2019-03-08	A második fejezet elkészült.	Nyitrai Dávid
0.3.0	2019-03-10	Az olvasónapló csatolása.	Nyitrai Dávid
0.4.0	2019-03-15	A harmadik fejezet elkészült.	Nyitrai Dávid

DRAFT

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.5.0	2019-03-22	A negyedik fejezet elkészült.	Nyitrai Dávid
0.5.0	2019-03-05	Az ötödik fejezet elkészült.	Nyitrai Dávid
0.6.0	2019-04-12	A hatodik fejezet elkészült.	Nyitrai Dávid
0.7.0	2019-04-26	A hetedik fejezet elkészült.	Nyitrai Dávid
0.8.0	2019-05-03	A nyolcadik fejezet elkészült.	Nyitrai Dávid
0.8.1	2019-05-08	Javítások, kiegészítések.	Nyitrai Dávid
1.0.0	2019-05-08	Első kiadás.	Nyitrai Dávid

DRAFT

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]

DRAFT

Tartalomjegyzék

I. Bevezetés 1

1. Vízió 2
1.1. Mi a programozás? . . . 2
1.2. Milyen doksikat olvassak el? . . . 2
1.3. Milyen filmeket nézzek meg? . . . 2

II. Tematikus feladatok 3

2. Helló, Turing! 5
2.1. Végtelen ciklus . . . 5
2.2. Lefagyott, nem fagyott, akkor most mi van? . . . 6
2.3. Változók értékének felcserélése . . . 8
2.4. Labdapattogás . . . 9
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . 12
2.6. Helló, Google! . . . 12
2.7. 100 éves a Brun tétel . . . 15
2.8. A Monty Hall probléma . . . 15

3. Helló, Chomsky! 18
3.1. Decimálisból unárisba átváltó Turing gép . . . 18
3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen . . . 19
3.3. Hivatkozási nyelv . . . 20
3.4. Saját lexikális elemző . . . 21
3.5. l33t.l . . . 22
3.6. A források olvasása . . . 24
3.7. Logikus . . . 27
3.8. Deklaráció . . . 28

DRAFT

4. Helló, Caesar! 35
4.1. double ** háromszögmátrix . . . 35
4.2. C EXOR titkosító . . . 36
4.3. Java EXOR titkosító . . . 37
4.4. C EXOR törő . . . 39
4.5. Neurális OR, AND és EXOR kapu . . . 42
4.6. Hiba-visszaterjesztéses perceptron . . . 44

5. Helló, Mandelbrot! 45
5.1. A Mandelbrot halmaz . . . 45
5.2. A Mandelbrot halmaz a `std::complex` osztállyal . . . 47
5.3. Biomorfok . . . 48
5.4. A Mandelbrot halmaz CUDA megvalósítása . . . 50
5.5. Mandelbrot nagyító és utazó C++ nyelven . . . 53
5.6. Mandelbrot nagyító és utazó Java nyelven . . . 53

6. Helló, Welch! 54
6.1. Első osztályom . . . 54
6.2. LZW . . . 57
6.3. Fabejárás . . . 60
6.4. Tag a gyökér . . . 63
6.5. Mutató a gyökér . . . 67
6.6. Mozgató szemantika . . . 72

7. Helló, Conway! 73
7.1. Hangyaszimulációk . . . 73
7.2. Java életjáték . . . 75
7.3. Qt C++ életjáték . . . 75
7.4. BrainB Benchmark . . . 75

8. Helló, Schwarzenegger! 77
8.1. Szoftmax Py MNIST . . . 77
8.2. Mély MNIST . . . 78
8.3. Minecraft-MALMÖ . . . 80

DRAFT

9. Helló, Chaitin! 82

9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . 82

9.2. Gimp Scheme Script-fu: króm effekt . . . 83

9.3. Gimp Scheme Script-fu: név mandala . . . 86

10. Helló, Olvasónapló! (Gutenberg) 91

10.1. Magas szintű programozási nyelvek 1 by Juhász István (Pici könyv) . . . 91

10.2. Magas szintű programozási nyelvek 2 by Juhász István (Pici könyv 2) . . . 95

10.3. A C programozási nyelv by Brian W. Kernighan – Dennis M. Ritchie (K and R könyv) . . . 95

10.4. Szoftverfejlesztés C++ nyelven by Benedek Zoltán - Levendovszky Tihamér (BME C++ könyv) . . . 96

III. Második felvonás 102

11. Helló, Arroway! 104

11.1. A BPP algoritmus Java megvalósítása . . . 104

11.2. Java osztályok a Pi-ben . . . 104

IV. Irodalomjegyzék 105

11.3. Általános . . . 106

11.4. C . . . 106

11.5. C++ . . . 106

11.6. Lisp . . . 106

DRAFT

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!

DRAFT

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.

DRAFT

I. rész

Bevezetés

DRAFT

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.

DRAFT

II. rész

Tematikus feladatok

DRAFT

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

A feladat megoldásában segített: Boros István.

Megoldás forrása: https://github.com/Matchbox1233/Turing/blob/master/1a.c

```
Végtelen ciklus egy magra 100% CPU:
    #include <stdio.h>

    int main()
    {

        int a=1;
        while (a=1)
        {

        }

        return 0;
    }
```

Megoldás forrása: https://github.com/Matchbox1233/Turing/blob/master/1b.c

```
Végtelen ciklus egy magra 0% CPU:
    #include <unistd.h>
    int main ()
    {
        for (;;)
         {
          sleep (1);
         }
        return 0;
    }
```

Megoldás forrása: https://github.com/Matchbox1233/Turing/blob/master/1c.c

```
Végtelen ciklus az összes magra 100% CPU:
    #include <stdio.h>
    #include <math.h>
    int main()
    {
        unsigned int a= pow(2,32)-1;
        #pragma omp parallel for schedule(dynamic, 1)
        for(unsigned int i=0;i<a;i++)
        {
            //printf("%d. Ez egy végtelen ciklus.\n",i);
            if(i>200000000) i=0;
        }
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

Egy olyan programot kell készítenünk, ami a végtelenségig fut. Ebben az esetben a "while" ciklus az egyik megoldás, de persze sok más módon is meg lehet oldani ezt a feladatot.

A "while" ciklus működése: Addig futattja le a ciklusmagban(a "{}" zárójel közötti rész) található kódot, amíg a feltétel(a "while" szó utáni zárójel) igaz.

Először is megadunk egy "int" változót ami jelen esetben egy. Ezután a "while" ciklust. A "while" feltételébe beleírjuk hogy az "a" változónk egyenlő eggyel. Mivel a ciklusmagban nem adtunk meg semmit, ezért a végtelenségig fog futni.

A program lefordításánál megadhatjuk, hogy hány magot terheljen a program, ezt a "-lpthread" paranccsal tehetjük meg.

A terminálban a "htop" paranccsal tudjuk ellenőrizni, hogy az egyes processzormagok hány százalékon dolgoznak.

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

```
Program T100
    {

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

DRAFT

```
      }

      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 fog kiírni erre a `T1000(T1000)` futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat:

A feladat a megállási problémát mutatja be. A megállási probléma: el lehet e dönteni egy programról, hogy végtelen ciklusba kerül e. 1936-ban bebizonyították, hogy nem lehet ilyen programot írni, amely eldönti, hogy az adott program végtelen ciklusba kerül e, ezt Alan Turing bizonyította be.

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://github.com/Matchbox1233/Turing/blob/master/3.c

```
Két változó felcserélése:
    #include <stdio.h>

    int main()
    {
        int a=3, b=2;

        printf("Eredeti: %i %i\n",a,b);

        a=a+b;
        b=a-b;
        a=a-b;

        printf("Csere: %i %i\n",a,b);

        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A változók cseréje segédváltozó nélkül inkább egy matematikai feladat, mint sem egy programozói.

Először is adjunk meg két int változót aminek az értéke jelen esetben mindegy(de könnyebb kis számokkal számolni az ellenőrzésnél). Én ezeket a változókat megjelenítettem a kijelzőn, ezt a "printf" paranccsal tehetjük ezt meg. Utána elkezdtem leírni magát az algoritmust, ami a következő:

a=a+b ez annyit jelent, hogy az "a" változónhoz hozzáadtuk a "b" változó értékét, tehát az "a" értéke 5 lett(5=3+2).

b=a-b itt a "b" változó értéke fog megváltozni, hiszen az "a" változó már 5, tehát a "b" értéke 3 lett(3=5-2).

a=a-b végül megcserélődik a változó, hiszen az "a" változó értéke 2 lett 3 helyett, mivel 2=5-3(az előzőek által kiadott értékek szerint).

2.4. Labdapattogás

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

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

Megoldás forrása: https://github.com/Matchbox1233/Turing/blob/master/4a.c

Labdapattogtatás "if"-kel:

```
    #include <stdio.h>
    #include <curses.h>
    #include <unistd.h>

    int main ( void )
    {
        WINDOW *ablak;
        ablak = initscr ();

        int x = 0;  ///x és az y tengely
        int y = 0;

        int xnov = 1; //ekkora meredekséggel fog pattogni
        int ynov = 1;

        int mx;  ///a terminál ablak nagysága
        int my;

        for ( ;; )
        {

            getmaxyx ( ablak, my , mx );

            mvprintw ( y, x, "O" );  //itt állíthatjuk át, hogy mi ↩
               pattogjon


            refresh ();
            usleep ( 100000 );  ///a pattogás gyorsaságát adja meg

            x = x + xnov;
            y = y + ynov;

            if ( x>=mx-1 )
            { // elerte-e a jobb oldalt?
                xnov = xnov * -1;
            }
            if ( x<=0 )
            { // elerte-e a bal oldalt?
                xnov = xnov * -1;
```

```
            }
            if ( y<=0 )
            { // elerte-e a tetejet?
                ynov = ynov * -1;
            }
            if ( y>=my-1 )
            { // elerte-e a aljat?
                ynov = ynov * -1;
            }

        }

        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A feladatot meg lehet oldani "if"-kel és "if"-ek nélkül is.

Először nézzük "if"-ek segítségével. Ahhoz, hogy a programunk helyesen leforduljon gépi nyelvre, be kell írnunk ezt egy terminálba, ami abban a mappában van megnyitva, ahol található a fenti kód. "sudo apt-get install libncurses5-dev" ez feltelepíti a cursese.h-t, amiből használunk pár dolgot. A fordításnál használni kell a sor végén a "-lncurses" kifejezést.

Kezdjünk bele!

I. Megadjuk az "x" és az "y" tengelyeket, így létrejön egy koordináta rendszer, ahol a labda fog pattogni.

II. Megadjuk, hogy mekkora meredekséggel pattogjon. Ezeket a számokat lehet változtatni, és akkor a pattogás iránya befolyásolható.

III. A "for" cikulsban megadjuk az "if"-eket, amik azt a célt szolgálják, hogy a labda az adott terminál-ablakon kívülre ne pattogjon. Tehát, mielőtt a labda kipattogna a terminálból(negatív tartomány), akkor a program megszorozza -1-el a az x vagy az y tengelyet és így ennek hatására újra a pozitív irányba halad a pattogás.

A labda alakját lehet változtatni, például "0" helyett lehet "+" is vagy akár szavak is lehetnek. A labda pattogásának a gyorsaságát is lehet állítani, ha növeljük a számot, akkor lassul a pattogás, ha viszont csökkentjük, akkor gyorsulni fog.

Megoldás forrása: https://github.com/Matchbox1233/Turing/blob/master/4b.c

A feladat megoldásában segített: Boros István.

```
Labdapattogtatás "if" nélkül:
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>

    char bitzero(char x)
    {
        int i;
        char bitt = x&0x1;
        for (i=0; i<8; i++)
```

DRAFT

```
        {
            bitt |= (x>>i)&1;
        }
        return 1-bitt;
    }

    void rajzol(char width, char height)
    {
        int i;
        /*magasság*/
        for (i=1; i<=height; i++)
        {
            printf("\n");
        }
        /*szélesség*/
        for (i=1; i<=width; i++)
        {
            printf(" ");
        }
        printf("O\n");
    }

    int main()
    {
        char x=1, y=1, vx=1, vy=1;
        while(1)
        {
            system("clear");
            vx-=2*bitzero(79-x);    //balra pattanjon
            vx+=2*bitzero(x);       //jobbra pattanjon
            vy-=2*bitzero(24-y);    //lefele
            vy+=2*bitzero(y);       //felfele
            x+=vx;
            y+=vy;
            rajzol(x,y);
            usleep(100000);
        }
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A kód három részből áll. Az első részben nullázunk egy változót. A második rész a "void". Ez a rész végzi a rajzolást. Itt állíthatjuk be, hogy mi pattogjon, ez lehet szöveg szám, vagy akármi. Az első "for" ciklus az "y" "tengely mentén lépteti a labdát(\n is mutatja), a második pedig az "x" tengely mentén. A harmadik rész pedig a fő rész a "main". Itt történik maga a pattogás, amit a "rajz" függvény kirajzol.

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!

A feladat megoldásában segített: Boros István.

Megoldás forrása: https://github.com/Matchbox1233/Turing/blob/master/5.c

Gépi szó:

```
    #include <stdio.h>
    #include <stdlib.h>

    int main()
    {
        int n=1;
        int x=1;
        while(x<<=1) n++;
        printf("A gépi szó hossza: %d\n",n);
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

Az "int" mérete 32 bit a legtöbb esetben. A printf parancs segítségével kiíratjuk a terminálba a gépi szó hosszát. A hosszúság kiszámítására a "while" cuklust használjuk.

2.6. Helló, Google!

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

Megoldás forrása: https://github.com/Matchbox1233/Turing/blob/master/6.c

A feladat megoldásában segített: Lovász Botond.

Page-Rank:

```
    #include <stdio.h>
    #include <math.h>

    void
    kiir (double tomb[], int db)
    {

        int i;

        for (i=0; i<db; ++i)
        {
            printf("%f\n",tomb[i]);
        }
    }
```

```
double
tavolsag (double PR[], double PRv[], int n)
{

    int i;
    double osszeg=0;

    for (i = 0; i < n; ++i)
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);

    return sqrt(osszeg);
}

void
pagerank(double T[4][4])
{
    double PR[4] = { 0.0, 0.0, 0.0, 0.0 }; //ebbe megy az eredmény
    double PRv[4] = { 1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0}; //ezzel
       szorzok

    int i, j;

    for(;;)
    {

        // ide jön a mátrix művelet

        for (i=0; i<4; i++)
        {
            PR[i]=0.0;
            for (j=0; j<4; j++)
            {
                PR[i] = PR[i] + T[i][j]*PRv[j];
            }
        }

            if (tavolsag(PR,PRv,4) < 0.0000000001)
                break;

        // ide meg az átpakolás PR-ből PRv-be

            for (i=0;i<4; i++){
                PRv[i]=PR[i];
            }
    }

    kiir (PR, 4);
}
```

```
int main (void)
{
    double L[4][4] =
    {
        {0.0,  0.0,      1.0/3.0,  0.0},
        {1.0,  1.0/2.0,  1.0/3.0,  1.0},
        {0.0,  1.0/2.0,  0.0,      0.0},
        {0.0,  0.0,      1.0/3.0,  0.0}
    };

    double L1[4][4] =
    {
        {0.0,  0.0,      1.0/3.0,  0.0},
        {1.0,  1.0/2.0,  1.0/3.0,  0.0},
        {0.0,  1.0/2.0,  0.0,      0.0},
        {0.0,  0.0,      1.0/3.0,  0.0}
    };

    double L2[4][4] =
    {
        {0.0,  0.0,      1.0/3.0,  0.0},
        {1.0,  1.0/2.0,  1.0/3.0,  0.0},
        {0.0,  1.0/2.0,  0.0,      0.0},
        {0.0,  0.0,      1.0/3.0,  1.0}
    };

    printf("\nAz eredeti mátrix értékeivel történő futás:\n");
    pagerank(L);

    printf("\nAmikor az egyik oldal semmire sem mutat:\n");
    pagerank(L1);

    printf("\nAmikor az egyik oldal csak magára mutat:\n");
    pagerank(L2);

    printf("\n");

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat:

Page-Rank:

A Page-Rank egy olyan algoritmus, amely hiperlinkekkel összekötött dokumentumokhoz számokat rendel azoknak a hiperlink-hálózatban betöltött szerepe alapján. Minden weboldalnak számolja a hivatkozásait, és az adott lapra történő hivatkozásokat is, így állít sorrendet, mert minél több a hivatkozás, annál többen nézik az oldalt. Így rendezi a találatokat a keresésnél.

A kód ezt az éréket számolja(tavolsag), és kiírtaja az eredményt.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

Megoldás forrása: https://github.com/Matchbox1233/Turing/blob/master/7.R

```
Brun tétel:
    library(matlab)

    stp <- function(x)
    {
        primes = primes(x)
        diff = primes[2:length(primes)]-primes[1:length(primes)-1]
        idx = which(diff==2)
        firstnumber = primes[idx]
        secondnumber = primes[idx]+2
        rec= 1/t1primes+1/t2primes
        return(sum(rec))
    }

    x=seq(13, 1000000, by=10000)
    y=sapply(x, FUN = stp)
    plot(x,y,type="b")
```

Tanulságok, tapasztalatok, magyarázat:

Brun tétel:

A Brun tétel prímszámokkal foglalkozik. A tétel szerint az ikerprímek(olyan prímek melyek különbsége 2) reciprokösszege egy véges értékhez konvergál. Viszont a sima prímszámok a végtelenségig folytatódnak.

A program elkészítéséhez szükség lesz egy számoló programra. Ez a program az "R-base" névre hallgat. Ha elindítjuk a programot szükség lesz még egy matlab nevű programra is, ami a megjelenítést fogja elvégezni. Ennek a telepítése az "r" terminálában zajlik. Telepítő parancs: "install.packages(matlab)". A parancs lefutása után kell még egy parancs: "library(matlab)".

Létrehozzuk az "stp" nevű algoritmust, ezzel tudunk majd hivatkozni a programunkra. Elsőnek egy vektorban(primes) eltároljuk az összes prímet(1.sor). Ezután eltoljuk a prímszámokat, úgy hogy a 2. elemtől kezdődjenek(primes[2:length(primes)]), és ebből kivonjuk azt az eltolást, amikor a prímszámoknak nem jelenítjük meg az utolsó elemét(primes[1:length(primes)-1]). A 3. sorban megvizsgáljuk a kivonást, ahol az eredmény 2 lesz, azt a helyet eltároljuk az "idx" nevű vektorban(így kapjuk meg az ikerprímeket). A 4.sor a számpárok első tagját, az 5.sor pedig a számpárok második tagját tartalmazzák. A 6.sorban a számoknak a reciprokát vesszük(elosztjuk őket eggyel) és ezt eltároljuk a "rec" nevű vektorban. A reciprokokat pedig összeadjuk az utolsó sorban a "sum" függvény segítségével.

A kód utolsó 3 sora pedig megjeleníti az ábrát, amit akkor kapunk ha lefutattjuk a programot.

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://github.com/Matchbox1233/Turing/blob/master/8.R

```
Monty Hall probléma:
    kiserletek_szama=10000000
    kiserlet = sample(1:3, kiserletek_szama, replace=T)  ///replace=T ==>  ↩
       ismétlődhet
    jatekos = sample(1:3, kiserletek_szama, replace=T)  ///sample ==>  ↩
       végtelen szám gen
    musorvezeto=vector(length = kiserletek_szama)

    for (i in 1:kiserletek_szama) {

        if(kiserlet[i]==jatekos[i]){

            mibol=setdiff(c(1,2,3), kiserlet[i])

        }else{

            mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))

        }

        musorvezeto[i] = mibol[sample(1:length(mibol),1)]

    }

    nemvaltoztatesnyer= which(kiserlet==jatekos)
    valtoztat=vector(length = kiserletek_szama)

    for (i in 1:kiserletek_szama) {

        holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
        valtoztat[i] = holvalt[sample(1:length(holvalt),1)]

    }

    valtoztatesnyer = which(kiserlet==valtoztat)


    sprintf("Kiserletek szama: %i", kiserletek_szama)
    length(nemvaltoztatesnyer)
    length(valtoztatesnyer)
    length(nemvaltoztatesnyer)/length(valtoztatesnyer)
    length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

Tanulságok, tapasztalatok, magyarázat:

Monty Hall probléma:

A Monty Hall-paradoxon egy valószínüségi paradoxon, ami az Amerikai Egyesült Államokban futott "Let's Make a Deal" (Kössünk üzletet) című televíziós vetélkedőben. A nevét a vetélkedő műsorvezetőjéről, Monty Hallról kapta.

A műsor végén mutatnak a játékosnak három ajtót. Kettő mögött egy-egy kecske, de a harmadik mögött egy új autó van. A játékos nyereménye az, ami az ő általa kiválasztott ajtó mögött van. Viszont a választás egy kicsit meg van bonyolítva. A játékos először csak rámutat az ajtóra mielőtt kinyitná, ezután a műsorveztő választ egy ajtót(a műsorvezető tudja, hogy melyik ajtó mögött mi van), majd megkérdezi a játékost, hogy szeretne e változtatni a döntésén. A játékos vagy változtat vagy nem , végül az ajtó kinyílik.

Elsőnek is megadjuk a kísérletek számát(első sor). A "kiserlet" nevű vektorban eltároljuk a nyertes ajtók számát(amik mögött kocsi van). A "jatekos" nevű vektorban meg eltároljukl, hogy a játékos melyik ajtót választotta. A "musorvezeto" vektorban eltároljuk a műsorvezető választását.

A forciklus a kísérletek számáig tart. Az "if"-ben azt határozzuk meg, ha a játékos első választásra eltalálja a nyereményt, akkor a műsorvezetőnek generálunk egy random számot(mert mindegy, hogy mit mond, hisz a játékos eltalálta a nyerő ajtót). Ha ez nem így van(else), akkor a három számból vegyük el azt a számot, amit a játékos választott, valamint azt is amit a műsorvezető választ. A második "for" ciklus azokat az eseteket vizsgálja, amikor a játékos változtat, és így kivesszük a három választásból a játékos, valamint a műsprvezető válaszát. MAjd megkpajuk azt is, amikor a játékos változtat, de nyer is. A végén meg kiírtajuk az eredményt.

DRAFT

3. fejezet

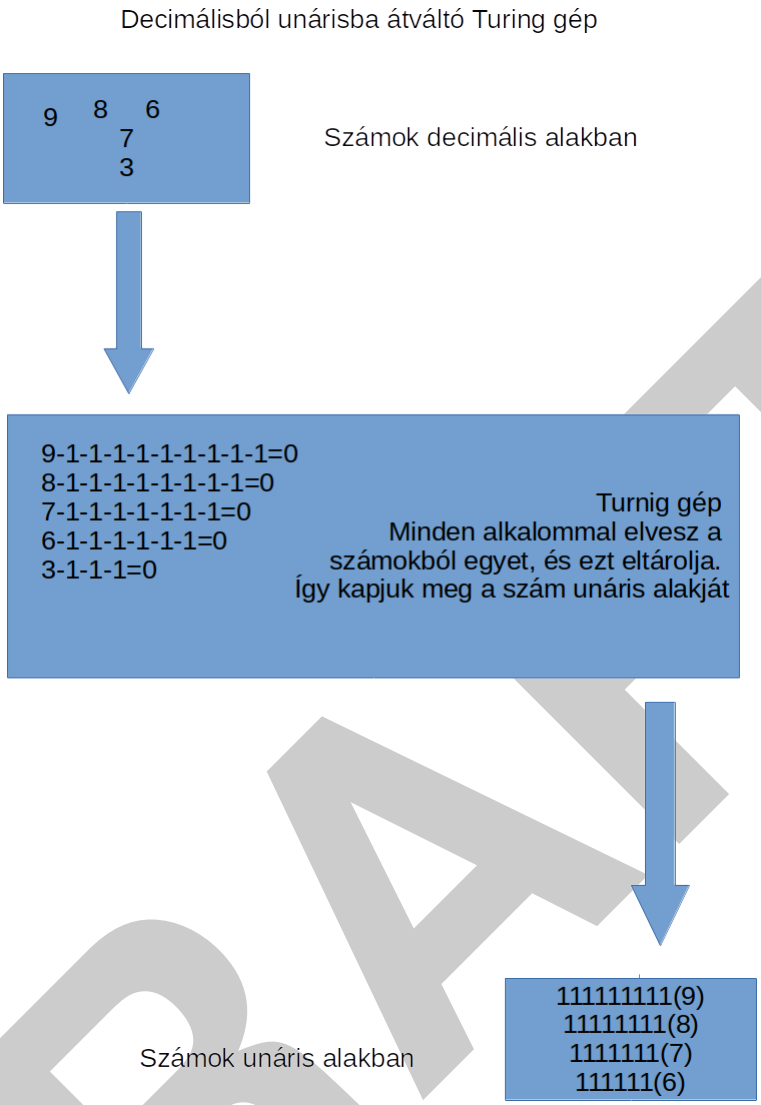
Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

A feladat megoldásában segített: Fürjes Benke Péter.

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

DRAFT



Tanulságok, tapasztalatok, magyarázat:

Turing gép: Alan Turing angol matematikus dolgozta ki 1936-ban. A gép az akkoriban még nem létező számítógépek működését modellezte. Három hardveres egységből állt: szalagtár(memória és input-output perifériák), vezérlőegységből(CPU), és az író olvasó fejből.

Az unáris számreednszer csak egyeseket tartalmaz(pl: 4=1111). A gép a decimális alakaban megadott számokat váltja át unárissá.

Az átváltás folyamata a következő. A gép minden egyes számot egyesével csökkent, addig amíg el nem éri a nullát,ezeket az egyeseket tárolja el a gép, és a végén ezt kapjuk eredményül.

A számok átváltását az utolsó számjegytől kezdjük, ha ez nulla, akkor kilenccel megyünk a kék állapotban, mindig levonunk egyet, addig, amíg el nem erjük a nullát. Annyi egyest tárolunk el, ahányszor a műveletet elvégeztük.

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

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

DRAFT

Megoldás forrása:

```
I. Környzetfüggő grammatika:
    Szabályok:
        S-->aBSc
        S-->abc
        Ba-->aB
        Bb-->bb

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

```
II. Környezetfüggő grammatika:
    Szabályok:
        S-->abc
        S-->aXbc
        Xb-->bX
        Xc-->Ybcc
        bY-->Yb
        aY-->aaX
        aY-->aa

    Példa:
        S --> aXbc --> abXc --> abYbcc --> aYbbcc --> aaxbbcc --> aabXbcc  ↩
           --> aabbXcc --> aabbYbccc --> aabYbbccc --> aaYbbbccc -->  ↩
           aaabbbccc
```

Tanulságok, tapasztalatok, magyarázat:

A generatív nyelvtan elméletetet és a Chomsky-hierarchiát Noam Chomsky dolgozta ki. A generatív nyelvelmélet célja: olyan modell létrhezása, amelynek a segítségével végtelen számú mondatot alkothatunk.

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:

```
C utasítás BNF-ben:

    <labeled-statement> ::= <identifier> : <statement>
                | case <constant-expression> : <statement>
                | default : <statement>

    <selection-statement> ::= if ( <expression> ) <statement>
                | if ( <expression> ) <statement> else <statement>
                | switch ( <expression> ) <statement>
```

```
<iteration-statement> ::= while ( <expression> ) <statement>
            | do <statement> while ( <expression> ) ;
            | for ( {<expression>}? ; {<expression>}? ; {<expression>}? ↩
                ) <statement>

<jump-statement> ::= goto <identifier> ;
            | continue ;
            | break ;
            | return {<expression>}? ;
```

Tanulságok, tapasztalatok, magyarázat:

Maga a BNF(Backus-Naur-forma) az a számítógépek programozási nyelveinek nyelvtanának leírása, ideértve az utasítás készleteket és a kommunikációs protokollokat is, valamint az egyes természetes nyelvek nyelvtanát. Számos programozási nyelv elméleti leírása és/vagy szemantikai dokumentumai általában BNF-ban vannak leírva.

A szabály bal oldala azt mutatja meg, hogy mivel/mikkel helytesíthető a jobb oldalon álló szimbólum. Az a szimbólum, amely soha nem bukkan fel szabály bal oldalán, az úgynevezett terminális.

Megoldás forrása:

```
C89 és C99 különbség:
    # include <stdio.h>
    int main()
    {
        long int a=987654321;
        for (int i = 0; i < a; ++i)
        {
            printf("Hello World!\n");
        }
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A program addig íja ki a képernyőre a "Hello World"-t, amíg a "long int" tart. A különbség a C89 és a C99 között, hogy még C89-ben nem fordulna le ez a program, mert ott még nem lehetett a "for" ciklus ciklusfejébe ciklusváltozót deklarálni, de a C99-ben már lehet.

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: hhttps://github.com/Matchbox1233/Chomsky/blob/master/4.c

```
Számokat számoló lexikális elemző:
    %{
    #include <stdio.h>
    int szamok=0;
    %}

    %option noyywrap

    %%
    [[:digit:]]+  //legalább egy számjegy egymás után
    {
        szamok++;
    }

    [[:alpha:]][[:print:]]*
    %%

    int main(void)
    {
        yylex();
        printf("%d számot talált a lexer: \n",szamok);
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A porgramhárom részből áll. Az első részben(%{-%}) létrehozunk egy "int" típusú változót "szamok" néven, amiben altároljuk a lexer által észlelt számokat. A második részben(%%-%%) Megadjuk a lexernek, hogy az egyes mintákra, hogyan reagáljon. Ha a "digit" legalább egy számjegyet talál egymás után, akkor eggyel növeli a számlálót, de ha nem talál ilyen számjegyet, akkor ne csináljon semmit. A harmadik részben(int main) meghívjuk a lexert, a "yylex" fügvénnyel, ami véigfutattja a bemenetet. Majd a program végén a "printf" -el kiíratjuk a számláló érékét.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás forrása: hhttps://github.com/Matchbox1233/Chomsky/blob/master/5.c

```
Leet cipher:
    %{
        #include <stdio.h>
        #include <ctype.h>
        #include <string.h>
        #include <time.h>
        #include <stdlib.h>
        int x=0;
        typedef struct{
            char c;
```

DRAFT

```
            char *d[6];
        } cipher;
        cipher L337[] = {
            {'a',{"4","/-\\","a","/\\","a","a"}},
            {'b',{"!3","|3","8","ß","b","b"}},
            {'c',{"[","<","{","c","c","c"}},
            {'d',{"|}","d","|>","T)","d","d"}},
            {'e',{"3","&","e","€","e","e"}},
            {'f',{"|=","|#","/=","f","f","f"}},
            {'g',{"&","6","g","(_+","g","g"}},
            {'h',{"|-|",")-(","[-]","h","h","h"}},
            {'i',{"1","[]","!","|","i","i"}},
            {'j',{"_|",";","1","j","j","j"}},
            {'k',{">|","1<","|c","k","k","k"}},
            {'l',{"1","|_","l","|","l","l"}},
            {'m',{"/\\/\\","/V\\","[V]","m","m","m"}},
            {'n',{"</>","n","|\\|","^/","n","n"}},
            {'o',{"0","Q","o","<>","o","o"}},
            {'p',{"|*","|>","p","|7","p","p"}},
            {'q',{"(_,)","q","9","&","q","q"}},
            {'r',{"I2","|?","Iz","r","r","r"}},
            {'s',{"s","5","z","§","s","s"}},
            {'t',{"4","-|-","7","t","t","t"}},
            {'u',{"(_)","u","v","L|","u","u"}},
            {'v',{"v","\\/","|/","\\|","v","v"}},
            {'w',{"\\/\\/","w","\\x/","\\\\//\\\\//","w","w"}},
            {'x',{"4","}{",">","<","x","x","x"}},
            {'y',{"y","j","`/","\\|/","y","y"}},
            {'z',{"2","-/_","z",">_","z","z"}},

            {'1',{"I","1","L","I"}},
            {'2',{"R","2","2","Z"}},
            {'3',{"E","3","E","3"}},
            {'4',{"4","A","A","4"}},
            {'5',{"S","5","S","5"}},
            {'6',{"b","6","G","6"}},
            {'7',{"7","7","L","T"}},
            {'8',{"8","B","8","B"}},
            {'9',{"g","q","9","9"}},
            {'0',{"0","()","[]","0"}},
        };

    %}
    %option noyywrap
    %%
    \n  {
            printf("\n");
        }
    .   {
            srand(time(0)+x++);
```

```
            char c = tolower(*yytext);

            int i=0;
            while(i<36 && L337[i++].c!=c);
            if(i<36)
            {
                char *s=L337[i-1].d[rand()%6];
                printf("%s",s);
            }
            else
            {
                printf("%c",c);
            }

        }
    %%

    int main()
    {
        yylex();
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

Mi is az a Leet?

A Leet lényege, hogy az írott szöveg betűit más számokkal és más ASCII karakterekkel helyettesítik. Ennek a célja a titkosítás volt. Az interenten zárt közösségek használták, hogy mások ne értsék, hogy ők miről bezélgetnek, írnak, stb.

A programot fel tudjuk osztani három főbb részre. Az első rész taralmazza az "include"-kat, egy "int" típusú változót, amely a random számok generálásához kell. Az első rész tartalmazz a program alapját, ez a chiper típusú tömb. Ebben a tömbben adjuk meg, hogy a számokat, és a betűket milyen karakterek helyetesítsék. Az esetek többségében három kódolt betűt, és négy eredeti betűt, hogy ne legyen minden betű átalakítva.

A másdoki nagyobb résznek a feldata, hogy a program átalakítja az eredti karaktereket, a megadottakra. Ezt a folyamatot, úgy végzi, hogy a lexer által beolvasott karaktereket megnézni a tömbben, ha egyezést talál, akkor random választ egyet a megadottakból, és kicsérli. Ha nem talál egyezést, akkor az eredeti karaktert meghagyja.

A harmadik rész(int main), ez a program fő része. Ebben a részben a program meghívja a lexert, az "yylex" fügvénnyel. Ha a program sikersen lefut, akkor nullát ad vissza az operációs rendszernek, ezzel jelzi, hogy sikeres volt a program lefutatása.

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://github.com/Matchbox1233/Chomsky/blob/master/6.c

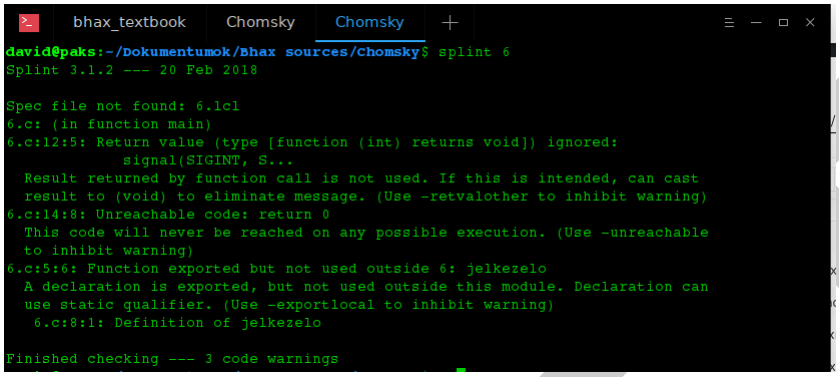
```
Eredeti program, ebbe kell beilleszteni a kódrészleteket:
    #include <stdio.h>
    #include <signal.h>


    void jelkezelo(int sig)
    {
        printf("Off %d\n",sig);
    }
    int main () {
    for(;;){
        if(signal(SIGINT, jelkezelo)==SIG_IGN)
           signal(SIGINT, SIG_IGN);
        }
    return 0;
    }
```

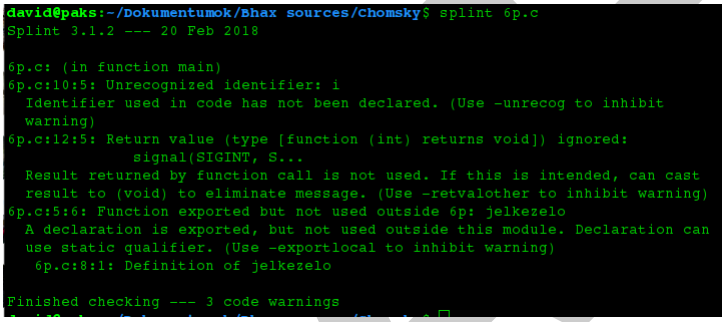
Tanulságok, tapasztalatok, magyarázat:

Az alap program magyarázata:

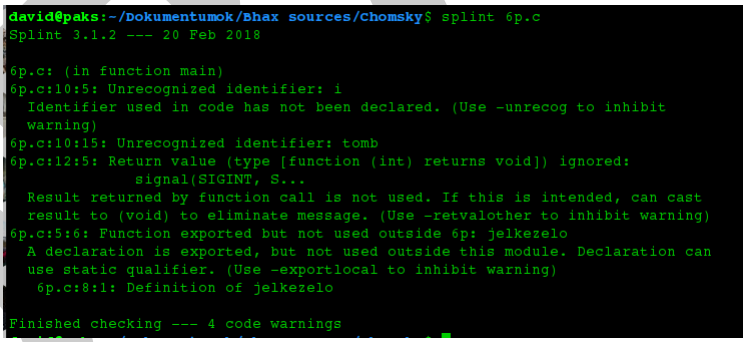
Olyan végtelen cklust késztettünk, amiből nem lehet kilépni, még a "ctrl+c" segítségével sem.



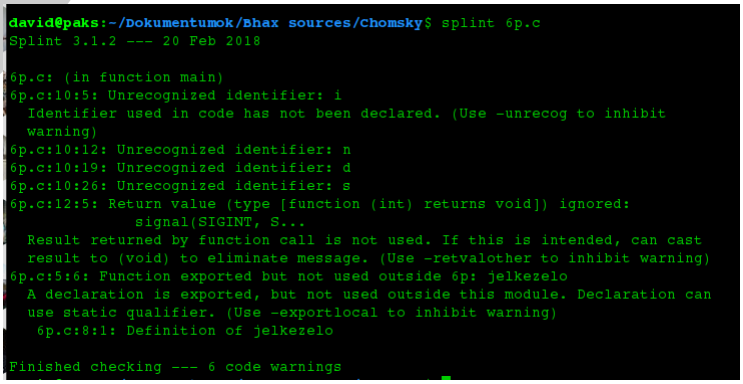
I.-II. A "for" ciklus addig fut, amíg az "i" kisebb, mint öt.



III. A "for" ciklus addig fog futni, amíg az "i" kisebb, mint öt, és a tömb minden elemét egyenlővé teszi "i"-vel



IV. A "for" ciklus addig fog futni, amíg "*d++" egynlő tud lenni az "s++"-al. Viszont ha az "n" értéke nagyobb, mint az "*s" vagy "*d" értéke, akkor a program nem fog működi.



V. A "printf" függvénnyel kiíratjuk először az "a+a+1"-t, a másodiknál pedig"(a+1)*2"

```
david@paks:~/Dokumentumok/bhax sources/chomsky$ splint 6p.c
Splint 3.1.2 --- 20 Feb 2018

6p.c: (in function main)
6p.c:10:5: Unrecognized identifier: i
  Identifier used in code has not been declared. (Use -unrecog to inhibit
  warning)
6p.c:10:12: Unrecognized identifier: n
6p.c:10:19: Unrecognized identifier: d
6p.c:10:26: Unrecognized identifier: s
6p.c:12:5: Return value (type [function (int) returns void]) ignored:
             signal(SIGINT, S...
  Result returned by function call is not used. If this is intended, can cast
  result to (void) to eliminate message. (Use -retvalother to inhibit warning)
6p.c:15:18: Unrecognized identifier: f
6p.c:15:20: Unrecognized identifier: a
6p.c:15:18: Argument 2 modifies <type <any>>, used by argument 3 (order of
    evaluation of actual parameters is undefined):
    printf("%d %d", f(a, ++a), f(++a, a))
  Code has unspecified behavior. Order of evaluation of function parameters or
  subexpressions is not defined, so if a value is used and modified in
  different places not separated by a sequence point constraining evaluation
  order, then the result of the expression is unspecified. (Use -evalorder to
  inhibit warning)
6p.c:15:29: Argument 3 modifies <type <any>>, used by argument 2 (order of
    evaluation of actual parameters is undefined):
    printf("%d %d", f(a, ++a), f(++a, a))
6p.c:5:6: Function exported but not used outside 6p: jelkezelo
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   6p.c:8:1: Definition of jelkezelo

Finished checking --- 10 code warnings
```

VI-VII. Először kiíratjuk, hogy az "f" milyen módon értékeli, ki az "a"-t, utána meg kiíratjuk az eredeti "a"-t. Ne írjunk olyan kódot, ami bármilyen értelemben feltételez egy kiértékelési sorrendet!

```
david@paks:~/Dokumentumok/bhax sources/chomsky$ splint 6p.c
Splint 3.1.2 --- 20 Feb 2018

6p.c: (in function main)
6p.c:10:5: Unrecognized identifier: i
  Identifier used in code has not been declared. (Use -unrecog to inhibit
  warning)
6p.c:10:12: Unrecognized identifier: n
6p.c:10:19: Unrecognized identifier: d
6p.c:10:26: Unrecognized identifier: s
6p.c:12:5: Return value (type [function (int) returns void]) ignored:
             signal(SIGINT, S...
  Result returned by function call is not used. If this is intended, can cast
  result to (void) to eliminate message. (Use -retvalother to inhibit warning)
6p.c:15:18: Unrecognized identifier: f
6p.c:15:20: Unrecognized identifier: a
6p.c:5:6: Function exported but not used outside 6p: jelkezelo
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   6p.c:8:1: Definition of jelkezelo

Finished checking --- 8 code warnings
```

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat:

I.jelentése: Minden számnál van nagyobb prímszám.

II.jelentése: Minden számnál van nagyobb iker prímszám pár.

III.jelentése: Minden prímszámnál van nagyobb prímszám.

IV.jelentése: Ha az egyik számnál nagyobb a másik, akkor az nem prím(ha ebben az esetben végesnek tekintjük a prímeket).

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;
  ```

DRAFT
  ```

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

A feladat megoldásában segített: Boros István.

Megoldás forrása: https://github.com/Matchbox1233/Chomsky/blob/master/8a.c

```
Egész:

    #include <stdio.h>

    int main()
    {
        int a;
        printf("%d\n",a);

        int b=10;
        printf("%d\n",b); //%d egész számot jelöl(decimális)

        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

Megadunk egy int típusú változót. Ha ennek nem adunk értéket, akkor nullát kapunk vissza, ha adnunk neki értéket, akkor azt az értéket kapjuk vissza kiíratáskor.

Megoldás forrása: https://github.com/Matchbox1233/Chomsky/blob/master/8b.c

```
Egészre mutató mutató:

    #include <stdio.h>

    int main()
    {
        int a=5;
        int *b = &a;

        printf("%d\n",*b);

        return 0;
    }
```
```

Tanulságok, tapasztalatok, magyarázat:

Megadunk egy int típusú "a" változót, amit eltárolunk "*b" pointerben(ez egy mutató). Így ha kiíratjuk "*b"-t akkor ötöt fogunk kapni, mert a "*b" öt lett, mivel az mutat az "a"-ra.

Megoldás forrása: https://github.com/Matchbox1233/Chomsky/blob/master/8c.cpp

Egész referenciája:

```
    #include <stdio.h>

    int main()
    {
        int a=5;
        int &r = a;

        printf("%d\n",r);

        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

Megadunk egy "a" változót, aminek az értéket öt. A következő sorban megadunk egy "r" opertátort, ami "a"-ra mutat, és így az "r" kiíratásánál megkapjuk az "a" értékét.

Megoldás forrása: https://github.com/Matchbox1233/Chomsky/blob/master/8d.c

Egészek tömbje:

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <time.h>

    int main()
    {
      int c[5];
      int i;
      srand(time(NULL));

      for(i=0;i<5;i++)
      {
        c[i]=rand()%100+1;
      }

      for(i=0;i<5;i++)
      {
        printf("%d\n",c[i]);
      }

      return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

Létrehozunk egy "int" típusú "c" nevű öt elemű tömböt. A "[]" zárójelbe írjuk, hogy mennyi elemet akarunk benne eltárolni. A kód többi része, annyit jelent, hogy ebbe a tömbbe generálunk random számokat 0-100 között(első "for" ciklus). Majd kiíratjuk őket(második "for" ciklus).

Megoldás forrása: https://github.com/Matchbox1233/Chomsky/blob/master/8e.cpp

```
Egészek tömbjének referenciája (nem az első elemé):

    #include <stdio.h>

    int main()
    {
      int a = 5;
      int &r = a;
      int c[5];
      int (&tr) [5] = c;
      printf("%d\n",&tr);

      return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A kód az előző kódokon alapszik. Csak itt egy kicsit bővül, azzal hogy a "tr" mutat a "c"-re. A kiíratáskor(printf) megkpajuk a tömb referebciaértékét.

Megoldás forrása: https://github.com/Matchbox1233/Chomsky/blob/master/8f.c

```
Egészre mutatók mutatók tömbje:

    #include <stdio.h>

    int main()
    {
      int *d[5];
      int c=**d;
      printf("%d %d\n",**d,c);

      return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A "*d" egy egészekre mutató tömb. Megadjuk a "c" változót, amiben eltároljuk a "*d" első elemét(ez a **d). Végül a "printf" függvény segítségével kiíratjuk az első elemét a "*d" tömbnek, és így ellenőrizzük, hogy a "c" is azt adja vissza.

Megoldás forrása: https://github.com/Matchbox1233/Chomsky/blob/master/8g.c

```
Egészre mutató mutató visszaadó függvény:

    #include <stdio.h>
```

```
    #include <stdlib.h>

    int *h ()
    {
      return (int*) malloc(sizeof(int));
    }

    int main()
    {
      printf("%p\n",h());

      return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

Létrehozunk egy mutató függvényt(*h), amiben a "malloc" lefoglal egy "int" méretű részt a memóriából(32 bit), amit a "return"-nek át kell konvertálnia "int*" típusra. A "main" részben pedig kiíratjuk a "printf" segítségével a lefoglalt memóriacímet.

Megoldás forrása: https://github.com/Matchbox1233/Chomsky/blob/master/8h.c

```
Egészre mutató mutató visszaadó függvényre mutató mutató:

    #include <stdio.h>
    #include <stdlib.h>

    int *h()
    {
        return (int*) malloc(sizeof(int));
    }

    int main()
    {
        int *(*l) () = h;
        printf("%p\n",l());
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

Az előző feladatra építkezve létrehozzuk az "l" mutató függvényt, amiben hivatkozunk a "h"-ra. Végül pedig kiíratjuk a "*h" függvényben lefoglalt memóriacímet.

Megoldás forrása: https://github.com/Matchbox1233/Chomsky/blob/master/8i.c

```
Egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, ↩
   egészet kapó függvény:

    #include <stdio.h>
    #include <stdlib.h>

    int sum(int a,int b)
```

```
    {
        return a+b;
    }

    int mul(int a, int b)
    {
        return a*b;
    }

    int (*asd(int c))()
    {
        if(c) return sum;
        else return mul;
    }

    int (*v(int c))()
    {
        return asd(c);
    }

    int main()
    {
        printf("%d\n",v(0)(5,5));
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

Létrehozunk egy szorzás(int mul) és egy összeg(int sum) függvényt. A kód úgy működik, hogy a 4. "int"-ben a "c"-t visszaküldjük az előtte lévő "int" -be(3. int), ott a program eldönti, hogy a "c" nulla e, mert ha nulla akkor a sorzáshoz, tér vissza az érték, ha pedig nullától különböző(lehet negatív is) akkor az összaadáshoz tér vissza a "c" értéke. Ezután megint visszatérünk a 4."int"-be, ahol a "*v" fog mutatni a "c"-re, és inenn fog tovább menni a "main"-be, ahol kiíratjuk az értékét.

Megoldás forrása: https://github.com/Matchbox1233/Chomsky/blob/master/8j.c

```
Függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató ↩
   mutatót visszaadó, egészet kapó függvényre:

    #include <stdio.h>
    #include <stdlib.h>

    int sum(int a,int b)
    {
        return a+b;
    }
    int mul(int a, int b)
    {
        return a*b;
    }
```

```
int (*asd(int c))()
{
    if(c) return sum;
    else return mul;
}

int (*v(int c))()
{
    return asd(c);
}

int main()
{
    int *(z(int))(int, int) = v;
    printf("%d\n",z(0)(5,5));
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat:

Létrehozunk egy szorzás(int mul) és egy összeg(int sum) függvényt. A kód úgy működik, hogy a 4. "int"-ben a "c"-t visszaküldjük az előtte lévő "int" -be(3. int), ott a program eldönti, hogy a "c" nulla e, mert ha nulla akkor a sorzáshoz, tér vissza az érték, ha pedig nullától különböző(lehet negatív is) akkor az összaadáshoz tér vissza a "c" értéke. Ezután megint visszatérünk a 4."int"-be, ahol a "*v" fog mutatni a "c"-re, és inenn fog tovább menni a "main"-be, ahol létrehozunk egy "*z"-t, ami a "v" értékére fog hivatkozni. Így a "printf"-ben a "z"-t fogjuk kiíratni.

DRAFT

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás forrása: https://github.com/Matchbox1233/Caesar/blob/master/1.c

Alsó háromszögmátrix:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num_rows = 5;
    double **tm = (double**) malloc(sizeof(double*)*num_rows);

    for(int i=0;i<num_rows;i++)
    {
        tm[i]=(double*) malloc(sizeof(double)*(i+1));
    }

    int k=0;

    for(int i=0;i<num_rows;i++)  //leképezés
    {
        for(int j=0;j<i+1;j++)
        {
            tm[i][j]=k++;
        }
    }
    for(int i=0;i<num_rows;i++)  //kiíratás
    {
        for(int j=0;j<i+1;j++)
        {
            printf("%lf ",tm[i][j]);
        }
        printf("\n");
```

```
        }
        for(int i=0;i<num_rows;i++)  //memóriacímek felszabadítása
        {
            free(tm[i]);
        }

        free(tm);
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A program elején megadjuk, hogy hány sora lesz a mátrixnak(int num_rows = 5;) A második sorban lefoglal a program 40 byte-ot a "malloc és sizeof" segítségével(malloc a sizeof-al 8byte-ot jelent az num_rows pedig 5byte-ot, és így lesz 40). Majd a "k" nullától minden elemnek 1.1-es leképezéssel adjuk a "k" -t. Utána kiíratjuk a terminálba az eredményt, végül felszabadítjuk a poniterekkel lefoglalt memóriacímeket.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: https://github.com/Matchbox1233/Caesar/blob/master/2.c

Exor titkosító:

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>

    int main(int argc, char *argv[])
    {
        if(argc > 2)
        {
            if(strlen(argv[2])!=8)
            {
                printf("Bad key\n");
                return -1;
            }
            FILE* fin = fopen(argv[1],"r");
            if(fin==NULL)
            {
                printf("File doesn't exist.\n");
                return -3;
            }
            else
            {
                char c;
                int i=0;
                while(!feof(fin))
```

```
                {
                    fscanf(fin,"%c",&c);
                    c^=argv[2][i%strlen(argv[2])];
                    printf("%c",c);
                }
                fclose(fin);
            }

        }
        else
        {
            printf("Használd: ./e fájlnév.txt kulcs\n");
            return -2;
        }
        printf("\n");
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A program először ellenőrzi, hogy megfelelő mennyiségű argumentumot kapott e(argv), ha nem akkor kiírja, hogy hogyan kell megadni az agrumentumokat. A program ellenőrzi, még hogy jó kulcsot kapott e a titkosításhoz, ha a kulcs nem áll nyolc karakterből, akkor kiírja, hogy: "Bad key".

Majd a harmadik "if"-es részben ellenőrizzük a titkosítandó fájlt. Ezt úgy tesszük meg, hogy a "fin" nevű változóban eltároljuk a megadott szöveget, ha ez egyenlő nullával, akkor a program kiírja, hogy: "File doesen't exist". Ellenkező esetben egy "while" ciklus beolvassa a szöveget, és a kulcs megfelelő karakterével össze exorozza(^), így titkosítja a szöveget. Miután titkosította kiírja a titkkosított szöveget a terminálba.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása: https://github.com/Matchbox1233/Caesar/blob/master/3.c

```
Exor titkosító java-ban:

    import java.io.*;
    import java.util.ArrayList;
    import java.util.Scanner;
    class exor
    {
        public static void main(String[] args)
        {
            FileInputStream fin=null;
            FileOutputStream fout=null;
            try
            {
                fin=new FileInputStream("tiszta.txt");
                int c;
```

DRAFT

```
                ArrayList<Integer> in_v = new ArrayList<Integer>();
                ArrayList<Integer> out_v = new ArrayList<Integer>();
                Scanner in = new Scanner(System.in);
                System.out.println("Enter key:");
                String key = in.nextLine();
                if(key.length() !=9)
                {
                    System.out.println("Bad key");
                    return;
                }
                while((c=fin.read())!=-1)
                {
                    in_v.add(c);
                }
                for(int i=0;i<in_v.size();i++)
                {
                    out_v.add(in_v.get(i)^key.charAt(i%9));
                }
                fout = new FileOutputStream("titkos.txt");
                for(int i=0;i<out_v.size();i++)
                {
                    System.out.printf("%c",out_v.get(i));
                }

            }
            catch(Exception ex)
            {

            }
            finally
            {
                try
                {
                    if(fin!=null) fin.close();
                    if(fout!=null) fout.close();
                }catch(Exception ex)
                {

                }
            }
        }
    }
```

Tanulságok, tapasztalatok, magyarázat:

Először is létrehozunk egy "class exor" nevű osztályt. Megadjuk, hogy a "tiszta.txt" -t tratalmát a "fin" nevű változóba töltse be. Utána beszekeneljük, ezt a szkennelést a "Scanner in" nevű változóban tároljuk el. Ezután bekérjük a kulcsot a felhasználótól(Enter key), és eltároljuk a "String key" nevű változóban. Ha a kulcs több mint nyolc karakterből áll, akkor a program kiírja, hogy "Bad key". Ha a kulcs megfelelt, akkor exor(^) segítségével elkzedődik a titkosítás, ez a "for" ciklus addig megy, amíg el nem éri a beolvasott fájl

végét(in_v.size). A titkosítás után a program kiírja a titkos szöveget a terminálba(titkos.txt).

4.4. C EXOR törő

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

Megoldás forrása: https://github.com/Matchbox1233/Caesar/blob/master/4.c

Exor törő:

```
    #define MAX_TITKOS 4096
    #define OLVASAS_BUFFER 256
    #define KULCS_MERET 8
    #define _GNU_SOURCE

    #include <stdio.h>
    #include <unistd.h>
    #include <string.h>

    double
    atlagos_szohossz (const char *titkos, int titkos_meret)
    {
        int sz = 0;
        for (int i = 0; i < titkos_meret; ++i)
            if (titkos[i] == ' ')
                ++sz;

        return (double) titkos_meret / sz;
    }

    int
    tiszta_lehet (const char *titkos, int titkos_meret)
    {
        // a tiszta szoveg valszeg tartalmazza a gyakori magyar szavakat
        // illetve az átlagos szóhossz vizsgálatával csökkentjük a
        // potenciális töréseket

        double szohossz = atlagos_szohossz (titkos, titkos_meret);

        return szohossz > 6.0 && szohossz < 9.0
               && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
               && strcasestr (titkos, "az") && strcasestr (titkos, "ha");

    }

    void
    exor (const char kulcs[], int kulcs_meret, char titkos[], int  ↩
       titkos_meret, char *buffer)
    {
```

```
    int kulcs_index = 0;

    for (int i = 0; i < titkos_meret; ++i)
    {

        buffer[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;

    }

}

void
exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
            int titkos_meret)
{

    char *buffer;

    if ((buffer = (char *)malloc(sizeof(char)*titkos_meret)) == NULL)
    {
        printf("Memoria (buffer) faliora\n");
        exit(-1);
    }

    exor (kulcs, kulcs_meret, titkos, titkos_meret, buffer);

    if(tiszta_lehet (buffer, titkos_meret))
    {
      printf("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
             kulcs[0],kulcs[1],kulcs[2],kulcs[3],kulcs[4],kulcs[5], ↩
                kulcs[6],kulcs[7], buffer);
    }

    free(buffer);

}

int main (void)
{
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;

    // titkos fajt berantasa
    while ((olvasott_bajtok =
                read (0, (void *) p,
                      (p - titkos + OLVASAS_BUFFER <
                       MAX_TITKOS) ? OLVASAS_BUFFER : titkos + ↩
```

```
                                MAX_TITKOS - p)))
            p += olvasott_bajtok;

        // maradek hely nullazasa a titkos bufferben
        for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
            titkos[p - titkos + i] = '\0';

        // osszes kulcs eloallitasa
        for (int ii = '0'; ii <= '9'; ++ii)
            for (int ji = '0'; ji <= '9'; ++ji)
                for (int ki = '0'; ki <= '9'; ++ki)
                    for (int li = '0'; li <= '9'; ++li)
                        for (int mi = '0'; mi <= '9'; ++mi)
                            for (int ni = '0'; ni <= '9'; ++ni)
                                for (int oi = '0'; oi <= '9'; ++oi)
                                    for (int pi = '0'; pi <= '9'; ++pi)
                                    {

                                        char *kulcs;

                                        if ((kulcs = (char *)malloc(sizeof( ↩
                                           char)*KULCS_MERET)) == NULL)
                                        {
                                            printf("Memoria (kulcs) faliora ↩
                                               \n");
                                            exit(-1);
                                        }

                                        kulcs[0] = ii;
                                        kulcs[1] = ji;
                                        kulcs[2] = ki;
                                        kulcs[3] = li;
                                        kulcs[4] = mi;
                                        kulcs[5] = ni;
                                        kulcs[6] = oi;
                                        kulcs[7] = pi;

                                        exor_tores (kulcs, KULCS_MERET,  ↩
                                           titkos, p - titkos);

                                        free(kulcs);
                                    }
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A program elő részében(atlagos_szohossz) a program megállpaítja, hogy mekkora a visszafejtett szavak átlagos hossza.

A második részben(tiszta_lehet) megvizsgálja a program, hogy visszafejtett szöveg tényleg lehet az, amit

a program visszafejtett. Ellenőrzi a szöveget, az átlagos szóhossz alapján, hogy az átlagosszóhossz hat és kilenc közés essen, és hogy tartalmazaz e a zöveg valamelyik szót a megadottakból: hogy, az, nem, ha.

A harmadik részben(exor) történik a visszafejtés az exorral(^) a megadott kulcsok segítségével.

A negyedik részbe(exor_tores) meghívjuk az "exor", és a "tiszta_lehet" függvényeket, ha ezeknek az értékei igazak, akkor a program kiírja a megoldást.

Az utolsó részben(int main) beolvassuk azt a szöveget, amit szeretnénk feltörni. A program megvizsgálja az összes kulcsot a nyolc egybeágyazott "for" ciklussal.

4.5. Neurális OR, AND és EXOR kapu

A feladatot R-ben készítsd el.

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

Megoldás forrása: https://github.com/Matchbox1233/Caesar/blob/master/5.R

A feladat megoldásában segítettem: Boros István.

```
Neurális OR, AND és EXOR kapu

    library(neuralnet)

    a1    <- c(0,1,0,1)
    a2    <- c(0,0,1,1)
    OR    <- c(0,1,1,1)

    or.data <- data.frame(a1, a2, OR)

    nn.or <- neuralnet(OR~a1+a2, or.data, hidden=0, linear.output=FALSE,  ↩
       stepmax = 1e+07, threshold = 0.000001)

    plot(nn.or)

    compute(nn.or, or.data[,1:2])


    a1    <- c(0,1,0,1)
    a2    <- c(0,0,1,1)
    OR    <- c(0,1,1,1)
    AND   <- c(0,0,0,1)

    orand.data <- data.frame(a1, a2, OR, AND)

    nn.orand <- neuralnet(OR+AND~a1+a2, orand.data, hidden=0, linear.output ↩
       =FALSE, stepmax = 1e+07, threshold = 0.000001)

    plot(nn.orand)

    compute(nn.orand, orand.data[,1:2])
```

```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR    <- c(0,1,1,0)

exor.data <- data.frame(a1, a2, EXOR)

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=0, linear.output= ↩
   FALSE, stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

compute(nn.exor, exor.data[,1:2])
```

```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR    <- c(0,1,1,0)

exor.data <- data.frame(a1, a2, EXOR)

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. ↩
   output=FALSE, stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

compute(nn.exor, exor.data[,1:2])
```

Tanulságok, tapasztalatok, magyarázat:

Betöltjük a neurális csomagot(library(neuralnet)). Az "a1"-be berakom a "0 1 0 1"-et, az "a2"-be pedig a "0 0 1 1"-et. Az "a1"-et és az "a2"-t logikai vagy művelettel megoldjuk, és így létrejön az "OR" tömb, amibe az eredmények kerülnek. Ezekből adatot csinálunk(or.data). A program megalkotja a neurális hálót, a "neuralnet" segístégével, és így beállítja a súlyokat.

A következőben, már az "a1"-et és az "a2"-t logikai és művelettel is megoldjuk, ennek az eredménye kerül az "AND" nevű tömbbe. A program ebből is adatot(orand.data), majd neurális hálót készít(neuralnet), az elkészített hálót hozzá illeszti az előzőhöz.

A harmadik részben exorozzuk az "a1" és az "a2" értékeit. Ott lesz az érték egy, ahol az értékek különböznek, például: ha "a1"-ben egy az érték, és "a2"-ben pedig kettő ugyanazon a helyen, akkor az exor értéke egy lesz. Ha "a1" és "a2" értéke azonos helyen ugyanannyi, akkor az exor értéke nulla lesz. A program itt is adatot készít, majd ebből neurális hálót. Ez az exoros rész különbözik az előzőektől, mert itt a hibák száma elég nagy. Ezt a következő részben oldjuk meg.

Az utolsó rész, megegyezik az előzővel, csak itt sokkal kevesebb hibát ad a program. Annyi a változás, hogy megadunk rejtett neuronaka(hidden=c(6, 4, 6)), amik segítségével a hibák száma csökken.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: https://github.com/Matchbox1233/Caesar/blob/master/6.cpp

```
Hiba - visszaterjesztési preceptron:

    #include "perceptron.hpp"
    #include <png++/png.hpp>

    using namespace std;

    int main(int argc, char *argv[])
    {
        if(argc!=2)
        {
            cout<<"Így használd: ./perc bemenet.png"<<endl;
            return -1;
        }
        png::image<png::rgb_pixel> png_img(argv[1]);
        int s = png_img.get_width()*png_img.get_height();

        Perceptron* p = new Perceptron(3,s,256,1);

        double* image = new double[s];

        for(int i=0;i<png_img.get_width();i++)
        {
            for(int j=0;j<png_img.get_height();j++)
            {
                image[i*png_img.get_width()+j]=png_img[j][i].red;
            }
        }

        double val = (*p)(image);
        cout<<val<<endl;
        delete [] image;
        delete p;
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A program először ellenőrzi az argumentumok számát(argc). Ha az argumentumok száma kettő, akkor a program tovbább fut, ha nem akkor kilép, és kiírja a termiinálba, hogy hogyan is futtasuk ezt a programot. Ezután beolvassa azt a fájlt, amit megadtunk neki, ebben az esetben ennek képnek kell lennie(.png). A következő sorban létrehozunk egy "perceptron"-t négy rétegű neurális hálóval, amibe eltároljuk a kép pixeleinek a számát például(256). Majd létrehozunk egy tömböt(double* image) amibe beletöltjük a kép vörös pixeleinek az értékét. A "for" ciklusban meghívjuk a "perceptront", és az feldolgozz a képet.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás forrása: https://github.com/Matchbox1233/Mandelbrot/blob/master/1.cpp

```
Mandelbort halmaz:

    #include <png++/png.hpp>

    #define N 500
    #define M 500
    #define MAXX 0.7
    #define MINX -2.0
    #define MAXY 1.35
    #define MINY -1.35

    void GeneratePNG( int tomb[N][M])
    {
        png::image< png::rgb_pixel > image(N, M);
        for (int x = 0; x < N; x++)
        {
            for (int y = 0; y < M; y++)
            {
                image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], tomb[x ↩
                    ][y]);
            }
        }
        image.write("kimenet.png");
    }

    struct Komplex
    {
        double re, im;
    };

    int main()
```

```
    {
        int tomb[N][M];

        int i, j, k;

        double dx = (MAXX - MINX) / N;
        double dy = (MAXY - MINY) / M;

        struct Komplex C, Z, Zuj;

        int iteracio;

        for (i = 0; i < M; i++)
        {
            for (j = 0; j < N; j++)
            {
                C.re = MINX + j * dx;
                C.im = MAXY - i * dy;

                Z.re = 0;
                Z.im = 0;
                iteracio = 0;

                while(Z.re * Z.re + Z.im * Z.im < 4 && iteracio++ < 255)
                {
                    Zuj.re = Z.re * Z.re - Z.im * Z.im + C.re;
                    Zuj.im = 2 * Z.re * Z.im + C.im;
                    Z.re = Zuj.re;
                    Z.im = Zuj.im;
                }

                tomb[i][j] = 256 - iteracio;
            }
        }

        GeneratePNG(tomb);

        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A program elsőrészében(void GeneratePNG) a program legenerálja a képet(png). Eltároljuk az "image[x][y]" -ban a kép RGB színeit pixelenként. A "void" végén kiíratjuk a képet.

A második rész a Komplex struktúra(struct Komplex). Itt meghatározzuk a komplex számok valós(re), és az immaginárius(im) részét.

A harmadik részben(int main) először létrehoznuk egy "int" típusú tömböt(int tomb), és három "int" típusú változót(i,j,k). Meghatározzuk, hogy milyen lépésközzel lépkedjen a komplex számokon(double dx, double dy). Utána létrehozunk "Komplex típusú" C, Z, Zuj nevű változókat, és egy "int" típus "iteracio" nevű

válltozót. A "for" ciklusban a Mandelbrot számítási képlete van megadva, ami ezen a linken megtalálható: https://hu.wikipedia.org/wiki/Mandelbrot-halmaz

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

Megoldás forrása: https://github.com/Matchbox1233/Mandelbrot/blob/master/2.cpp

```
Mandelbort halmaz std::complex osztállyal:

    #include <png++/png.hpp>
    #include <complex>

    const int N = 500;
    const int M = 500;
    const double MAXX = 0.7;
    const double MINX = -2.0;
    const double MAXY = 1.35;
    const double MINY = -1.35;

    void GeneratePNG(const int tomb[N][M])
    {
        png::image< png::rgb_pixel > image(N, M);
        for (int x = 0; x < N; x++)
        {
            for (int y = 0; y < M; y++)
            {
                image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], tomb[x ↩
                    ][y]);
            }
        }
        image.write("kimenet.png");
    }

    int main()
    {
        int tomb[N][M];

        double dx = (MAXX - MINX) / N;
        double dy = (MAXY - MINY) / M;

        std::complex<double> C, Z, Zuj;

        int iteracio;

        for (int i = 0; i < M; i++)
        {
            for (int j = 0; j < N; j++)
            {
                C.real (MINX + j * dx);
```

```
                C.imag (MAXY - i * dy);

                Z = 0;
                iteracio = 0;

                while(abs(Z) < 2 && iteracio++ < 255)
                {
                    Zuj = Z*Z+C;
                    Z = Zuj;
                }

                tomb[i][j] = 256 - iteracio;
            }
        }

        GeneratePNG(tomb);

        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A program annyiban különbözik az előzőtől, hogy itt nem készítünk előre Komplex struktúrát(struct Komplex), hanem a C++-ba beleépített komplex számokat használjuk, ezt úgy tehetjük meg, hogy az elején "include"-ba belerakjuk a "complex" szót, majd amikor használjuk, akkor csak elkészítjük az "std::complex" osztályt.

5.3. Biomorfok

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

Megoldás forrása: https://github.com/Matchbox1233/Mandelbrot/blob/master/3.cpp

```
Biomorf:

    #include <iostream>
    #include "png++/png.hpp"
    #include <complex>

    int
    main ( int argc, char *argv[] )
    {

        int szelesseg = 1920;
        int magassag = 1080;
        int iteraciosHatar = 255;
        double xmin = -1.9;
        double xmax = 0.7;
        double ymin = -1.3;
        double ymax = 1.3;
```

```
        double reC = .285, imC = 0;
        double R = 10.0;

        if ( argc == 12 )
        {
            szelesseg = atoi ( argv[2] );
            magassag =  atoi ( argv[3] );
            iteraciosHatar =  atoi ( argv[4] );
            xmin = atof ( argv[5] );
            xmax = atof ( argv[6] );
            ymin = atof ( argv[7] );
            ymax = atof ( argv[8] );
            reC = atof ( argv[9] );
            imC = atof ( argv[10] );
            R = atof ( argv[11] );

        }
        else
        {
            std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a ↩
                b c d reC imC R" << std::endl;
            return -1;
        }

        png::image < png::rgb_pixel > kep ( szelesseg, magassag );

        double dx = ( xmax - xmin ) / szelesseg;
        double dy = ( ymax - ymin ) / magassag;

        std::complex<double> cc ( reC, imC );

        std::cout << "Szamitas\n";


        for ( int y = 0; y < magassag; ++y )
        {

            for ( int x = 0; x < szelesseg; ++x )
            {

                double reZ = xmin + x * dx;
                double imZ = ymax - y * dy;
                std::complex<double> z_n ( reZ, imZ );

                int iteracio = 0;
                for (int i=0; i < iteraciosHatar; ++i)
                {

                    z_n = std::pow(z_n, 3) + cc;
                    //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
```

```
                    if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
                    {
                        iteracio = i;
                        break;
                    }
                }

                kep.set_pixel ( x, y,
                                png::rgb_pixel ( (iteracio*20)%255, ( ↩
                                   iteracio*40)%255, (iteracio*60)%255 )); ↩
                                   /színek
            }

            int szazalek = ( double ) y / ( double ) magassag * 100.0;
            std::cout << "\r" << szazalek << "%" << std::flush;
        }

        kep.write ( argv[1] );
        std::cout << "\r" << argv[1] << " mentve." << std::endl;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A program működéséhez elször megadjuk a szükséges osztályokat, függvényeket, et jelzi az "#include" szó. Az "int main" -ben először megadjuk a kép méreteit. A program futtatásához meg kell adnunk tizenkét argumentumot, ezt jelzni az "if (argc == 12)", ha nincs meg ez a tizenkettő, akkor a program kiírja, hogy hány argumentumot kell beírni, ezt jelzi az "else". Az "else" után eltároljuk a kép adatait a "png::image" változóban. Aztán meghatározzuk a lépésközt az x és az y tengelyeken(double dx, double dy). A lépésköz meghatározás után hasznákuk a komplex osztályt(std::complex). Majd a képernyőre kiírjuk a számítás eredményét. A második "for" ciklusban a pogram lpéked a komplex számokon(double reZ,double imZ). A harmadik "for" ciklusban a program megad egy értéket, ami alapján meghatározz a színeket, azztán ebből a "for" ciklusból ki is lép a "break" parancs segítségével. A "kep.set_pixel" RGB szerint meghatározz a színeket. Folyamatosan figyelhetjük, hogy a program hány százaléknál jár az "int szazalek" segítségével. A prgram végén pedig fájlba kiírjuk az eredményt.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás forrása: https://github.com/Matchbox1233/Mandelbrot/blob/master/4.cpp

```
Mandelbort halmaz CUDA megvalósításával:

    #include <png++/image.hpp>
    #include <png++/rgb_pixel.hpp>

    #include <sys/times.h>
    #include <iostream>


    #define MERET 600
```

DRAFT

```
#define ITER_HAT 32000

__device__ int mandel (int k, int j)
{
  float a = -2.0, b = .7, c = -1.35, d = 1.35;
  int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;

  float dx = (b - a) / szelesseg;
  float dy = (d - c) / magassag;
  float reC, imC, reZ, imZ, ujreZ, ujimZ;
  int iteracio = 0;

  reC = a + k * dx;
  imC = d - j * dy;

  reZ = 0.0;
  imZ = 0.0;
  iteracio = 0;

  while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
    {
      ujreZ = reZ * reZ - imZ * imZ + reC;
      ujimZ = 2 * reZ * imZ + imC;
      reZ = ujreZ;
      imZ = ujimZ;

      ++iteracio;

    }

  return iteracio;
}

__global__ void mandelkernel (int *kepadat)
{

  int tj = threadIdx.x;
  int tk = threadIdx.y;

  int j = blockIdx.x * 10 + tj;
  int k = blockIdx.y * 10 + tk;

  kepadat[j + k * MERET] = mandel (j, k);

}

void cudamandel (int kepadat[MERET][MERET])
{

  int *device_kepadat;
```

```
  cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));

  dim3 grid (MERET / 10, MERET / 10);
  dim3 tgrid (10, 10);
  mandelkernel <<< grid, tgrid >>> (device_kepadat);

  cudaMemcpy (kepadat, device_kepadat,
          MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
  cudaFree (device_kepadat);

}

int main (int argc, char *argv[])
{

  clock_t delta = clock ();

  struct tms tmsbuf1, tmsbuf2;
  times (&tmsbuf1);

  if (argc != 2)
    {
      std::cout << "Hasznalat: ./mandelpngc fajlnev";
      return -1;
    }

  int kepadat[MERET][MERET];

  cudamandel (kepadat);

  png::image < png::rgb_pixel > kep (MERET, MERET);

  for (int j = 0; j < MERET; ++j)
    {
      for (int k = 0; k < MERET; ++k)
    {
      kep.set_pixel (k, j,
             png::rgb_pixel (255 -
                     (255 * kepadat[j][k]) / ITER_HAT,
                     255 -
                     (255 * kepadat[j][k]) / ITER_HAT,
                     255 -
                     (255 * kepadat[j][k]) / ITER_HAT));
    }
    }
  kep.write (argv[1]);

  std::cout << argv[1] << " mentve" << std::endl;

  times (&tmsbuf2);
```

```
    std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
      + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;

    delta = clock () - delta;
    std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;

  }
```

Tanulságok, tapasztalatok, magyarázat:

A program első részében(__device__ int mandel (int k, int j)) történik meg a Mandelbrot halmaz kiszámítása. A "device" függvény csak videokártyáról érhető el, a "global" függvény, pedig elérhetővé teszi a "device"-ot.

A második részben(__global__ void mandelkernel (int *kepadat)) a program megnézi, hogy melyik CORE hajtja végre a programot, és ennek az idexét elküldi az első résznek, mert ez alapján számolja ki a halmazt.

A harmadik részben(void cudamandel (int kepadat[MERET][MERET])) lefoglalajuk a kártyán a helyet, és találkozunk egy nem tipikus fuggvényhívással, amiben megadjuk, hogy hány blokkot és hány szálat szánunk a programnak. A rész végén felszabadítjuk a pointereket.

Az utolsó rész a fő rész(int main). Itt először létrehozunk egy "clock_t delta" nevű változót, ezt arra fogjuk használni, hogy a pogram végén kiírja, hogy mennyi ideig futott. Ha nem jól indítanánk a programot, akkor az figyelmeztet minket. Erre szolgál az "if (argc != 2)", ami azt jelenti, hogy ha a bemeneti argumentum nem kettő, akkor kiírja, ami az "if"-ben van megadva. A "for" ciklusokkal megalkotjuk a képet, az RGB színekkel. A program a futása végén kiírja, hogy mentve, és kiírja a futási időt is.

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása: https://github.com/Matchbox1233/Mandelbrot/blob/master/5.cpp

Tanulságok, tapasztalatok, magyarázat:

A mandelbrot halmaz számolását a "frakszál" végzi, viszont a rajzolást átadja "frakablak" osztálynak, mindig az éppen kiszámolt sort. A main függvényben példányosítunk, majd a "w.show()"" segítségével kirajzoljuk a halmazt. A nagyítás a "frakablak.cpp-ben" történik. A "mousePressEvent" a nagyítandó terület bal felső sarkát határozza meg. A "mouseMoveEvent" a nagyítandó terület szélességét és magasságát határozza meg. A "mouseReleaseEvent" végzi el a nagyítást.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása: https://gitlab.com/fupn26/bhax/tree/master/attention_raising/Mandelbrot/Zoom_java

DRAFT

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.

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!

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/1a.java

Polártranszformáció Java-ban:

```
    import java.util.Random;

    public class PolarGen
    {
        private double tarolt;
        private boolean nincsTarolt;
        private Random r;
        private int RAND_MAX;

        public PolarGen()
        {
            nincsTarolt = true;
            r = new Random();
            r.setSeed(20);
            this.RAND_MAX=100;
        }
        public PolarGen(Integer RAND_MAX)
        {
            nincsTarolt = true;
            r = new Random();
            r.setSeed(20);
            this.RAND_MAX=RAND_MAX;
```

DRAFT

```
        }

        public double kovetkezo()
        {
            if (nincsTarolt)
            {

                double u1, u2, v1, v2, w;
                int i=0;
                do
                {
                      u1 = r.nextInt() / (RAND_MAX + 1.0);
                      u2 = r.nextInt() / (RAND_MAX + 1.0);
                      v1 = 2 * u1 - 1;
                      v2 = 2 * u2 - 1;
                      w = v1 * v1 + v2 * v2;
                }
                while (w > 1 && i++ < 40000000);
                double r = Math.sqrt ((2 * Math.log10(w)) / w);
                tarolt = r * v2;
                nincsTarolt = !nincsTarolt;
                return r * v1;
            }
            else
            {
                nincsTarolt = !nincsTarolt;
                return tarolt;
            }
        }
    }
```

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/1b.cpp

```
Polártranszformáció C++-ban:

    #include <iostream>
    #include <cstdlib>
    #include <cmath>
    #include <ctime>

    class PolarGen
    {
    public:
      PolarGen ()
      {
        nincsTarolt = true;
        std::srand (std::time (NULL));
      }
       ~PolarGen ()
      {
      }
```

```
    double kovetkezo ()
    {
      if (nincsTarolt)
        {
          double u1, u2, v1, v2, w;
          do
      {
        u1 = std::rand () / (RAND_MAX + 1.0);
        u2 = std::rand () / (RAND_MAX + 1.0);
        v1 = 2 * u1 - 1;
        v2 = 2 * u2 - 1;
        w = v1 * v1 + v2 * v2;
      }
          while (w > 1);

          double r = std::sqrt ((-2 * std::log (w)) / w);

          tarolt = r * v2;
          nincsTarolt = !nincsTarolt;

          return r * v1;
        }
      else
        {
          nincsTarolt = !nincsTarolt;
          return tarolt;
         }
     }



  private:
    bool nincsTarolt;
    double tarolt;

  };


  int
  main (int argc, char **argv)
  {
    PolarGen pg;

    for (int i = 0; i < 10; ++i)
      std::cout << pg.kovetkezo () << std::endl;

    return 0;
  }
```

Tanulságok, tapasztalatok, magyarázat:

A porgram elején megadjuk az include-okat. A program két nagyobb részből áll: az egyik ilyen rész a polargen osztály(class PolarGen) a másik rész pedig a "main". A polaregen osztálynak van egy "private" és egy "public" része. A "public" részben történik maga a számolás. A "private" részben pedig megadtunk két újabb változót, amiket a "public" részben használunk. A "main" részben pedig kiíratjuk a terminálba a számolással megkapott értékeket.

6.2. LZW

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

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/2.c

A megoldásban segített: Petrus József Tamás

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
typedef struct node{
    char c;
    struct node* left;
    struct node* right;
} Node;

Node* fa;
Node gyoker;

#define null NULL

Node* create_empty()
{
    Node* tmp = &gyoker;
    tmp->c= '/';
    tmp->left = null;
    tmp->right = null;
    return tmp;
}

Node* create_node(char val)
{
    Node* tmp = (Node*)malloc(sizeof(Node));
    tmp->c=val;
    tmp->left = null;
    tmp->right = null;
    return tmp;
}
```

```
void insert_tree(char val)
{
    if(val=='0')
    {
        if(fa->left == null)
        {
            fa->left = create_node(val);
            fa = &gyoker;
            //printf("Inserted into left.");
        }
        else
        {
            fa = fa->left;
        }
    }
    else
    {
        if(fa->right == null)
        {
            fa->right = create_node(val);
            fa = &gyoker;
            //printf("Inserted into left.");
        }
        else
        {
            fa = fa->right;
        }
    }
}

void inorder(Node* elem,int depth)
{

    if(elem==null)
    {
        return;
    }
    inorder(elem->left,depth+1);
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth]='\0';

        printf("%s%c\n",spaces,elem->c);
```

```
            free (spaces);
        }
        else
        {
            printf("%c\n",elem->c);
        }
        inorder(elem->right,depth+1);
    }

    void destroy_tree(Node* elem)
    {
        if(elem==null)
        {
            return;
        }
        destroy_tree(elem->left);
        destroy_tree(elem->right);
        if(elem->c == gyoker.c)
        {

        }
        else
        {
            free(elem);
        }
    }

    int main(int argc, char** argv)
    {
        srand(time(null));
        fa = create_empty();
        for(int i=0;i<100;i++)
        {
            int x=rand()%2;
            if(x)
            {
                insert_tree('1');
            }
            else
            {
                insert_tree('0');
            }
        }

        inorder(&gyoker,0);

        destroy_tree(&gyoker);
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A program sok részből áll. Először is megadjuk az "include-okat", majd egy rövídítést is(typedef struct). Az utána lévő sorban definiálunk egy "NULL"-t "null"-ra, ez csak kis írásbeli könnyítés. Az első osztályban(Node* create_empty()) megadjuk a fa gyökerének a kezdőértékét.A "tmp" az a fa mutató, tmp csak a függvényen belül fog létezni. A kitüntetett gyökérelem a "/" jel, és értéket kell adani a "tmp"-nek, ez az érték nulla lesz. A második osztályban(ode* create_node(char val)), ugyanaz történik, mint az elsőben, csak itt a fa ágainak a kezdőértékét adjuk meg. A harmadik osztályban(void insert_tree(char val)) eldől, hogy melyik érték megy a jobb, illetve a bal oldalra. A negyedik osztályban(void inorder(Node* elem,int depth)) egy fa bejárást láthatunk, ez egy "inorder" bejárás, itt először a fa bal oldalát, majd a fa gyökerét, és végül a fa jobb oldalát járjuk be. Az osztály első részében megadunk egy kilépési feltételt(return;). Ebben a részben megjelenik a "depth" nevű változó, ez a mélységért felel. Minél mélyebben van, annál jobbrább van a terminálban. A negyedik részben még létrehozunk egy "spaces" nevű pointert, ez felel a "-" kirajzolásáért a terminálban, ezt a pointert az osztály végén felszabadítjuk a "free" parancs segítségével. Az ötödik oasztály(void destroy_tree(Node* elem)) postorder módon járja be a fát és így szabadítja fel a pointereket. Az utolsó rész(int main(int argc, char** argv)) maga a fő rész. Itt a program eldönti, a nullás, és az egyes gyermekek helyét. Majd az "main" végén megint felszabadítjuk a pointert(estroy_tree).

6.3. Fabejárás

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

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/3.c

```
void inorder(Node* elem,int depth)
{

    if(elem==null)
    {
        return;
    }
    inorder(elem->left,depth+1);
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth]='\0';

        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
```

```
        }
        inorder(elem->right,depth+1);
    }

    void preorder(Node* elem,int depth)
    {
        if(elem==null)
        {
            return;
        }
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
                spaces[i+1]='-';
            }
            spaces[depth*2]='\0';

            printf("%s%c\n",spaces,elem->c);
        }
        else
        {
            printf("%c\n",elem->c);
        }
        preorder(elem->left,depth+1);
        preorder(elem->right,depth+1);
    }

    void postorder(Node* elem,int depth)
    {
        if(elem==null)
        {
            return;
        }
        postorder(elem->left,depth+1);
        postorder(elem->right,depth+1);
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
                spaces[i+1]='-';
            }
            spaces[depth*2]='\0';
```

```
            printf("%s%c\n",spaces,elem->c);
        }
        else
        {
            printf("%c\n",elem->c);
        }
    }

    int main(int argc, char** argv)
    {
        srand(time(null));
        fa = create_empty();
        gyoker = *fa;
        for(int i=0;i<100;i++)
        {
            int x=rand()%2;
            if(x)
            {
                insert_tree('1');
            }
            else
            {
                insert_tree('0');
            }
        }
        if(argc == 2)
        {
            if(strcmp(argv[1],"--preorder")==0)
            {
                preorder(&gyoker,0);
            }
            else if(strcmp(argv[1],"--inorder")==0)
            {
                inorder(&gyoker,0);
            }
            else if(strcmp(argv[1],"--postorder")==0)
            {
                postorder(&gyoker,0);
            }
            else
            {
                usage();
            }
        }
        else
        {
            usage();
        }
        destroy_tree(&gyoker);
        return 0;
```

```
    }
```

Tanulságok, tapasztalatok, magyarázat:

Az előző programot vesszük alapul. Csak kiegészítjük, még két osztállyal. Az egyik a "void preorder(Node* elem,int depth)" osztály, ez az osztály felelős a preorder bejárásért, ez a bejárási módszer először megvizsgálja a fa gyökerét, majd a bal aztán a jobb oldalt. A másik pedig a "void postorder(Node* elem,int depth)" osztály, ez pedig felelős a postorder bejárásért, ami azt jelenti, hogy először a fa baol oldalát, majd a jobb oldalát, és végül a fa gyökerét járjuk be. Ezeket a bejárások beépítettük a "main" függvénybe is.

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 forrása: https://github.com/Matchbox1233/Welch/blob/master/4.cpp

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string.h>

#define null NULL

class Binfa
{
private:
    class Node
    {
    public:
        Node(char c='/')
        {
            this->c=c;
            this->left = null;
            this->right = null;
        }
        char c;
        Node* left;
        Node* right;
    };
    Node* fa;


public:
    Binfa(): fa(&gyoker)
    {

    }
```

```
void operator<<(char c)
{
    if(c=='0')
    {
        if(fa->left == null)
        {
            fa->left = new Node('0');
            fa = &gyoker;
        }
        else
        {
            fa = fa->left;
        }
    }
    else
    {
        if(fa->right == null)
        {
            fa->right = new Node('1');
            fa = &gyoker;
        }
        else
        {
            fa = fa->right;
        }
    }
}

void preorder(Node* elem,int depth=0)
{
    if(elem==null)
    {
        return;
    }
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
```

```
            printf("%c\n",elem->c);
        }
        preorder(elem->left,depth+1);
        preorder(elem->right,depth+1);
    }

    void inorder(Node* elem,int depth=0)
    {
        if(elem==null)
        {
            return;
        }
        inorder(elem->left,depth+1);
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
                spaces[i+1]='-';
            }
            spaces[depth*2]='\0';

            printf("%s%c\n",spaces,elem->c);
        }
        else
        {
            printf("%c\n",elem->c);
        }
        inorder(elem->right,depth+1);
    }

    void postorder(Node* elem,int depth=0)
    {
        if(elem==null)
        {
            return;
        }
        postorder(elem->left,depth+1);
        postorder(elem->right,depth+1);
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
                spaces[i+1]='-';
            }
```

```
                spaces[depth*2]='\0';

                printf("%s%c\n",spaces,elem->c);
            }
            else
            {
                printf("%c\n",elem->c);
            }
        }

        void destroy_tree(Node* elem)
        {
            if(elem==null)
            {
                return;
            }
            destroy_tree(elem->left);
            destroy_tree(elem->right);
            if(elem->c!='/') delete elem;
        }

        Node gyoker;

    };

    void usage()
    {
        printf("Használat: ./binfa KAPCSOLÓ\n");
        printf("Az KAPCSOLÓ lehet:\n");
        printf("--preorder\tA bináris fa preorder bejárása\n");
        printf("--inorder\tA bináris fa inorder bejárása\n");
        printf("--postorder\tA bináris fa postorder bejárása\n");
    }

    int main(int argc, char** argv)
    {
        srand(time(0));
        Binfa bfa;
        for(int i=0;i<100;i++)
        {
            int x=rand()%2;
            if(x)
            {
                bfa<<'1';
            }
            else
            {
                bfa<<'0';
            }
        }
```

```
        if(argc == 2)
        {
            if(strcmp(argv[1],"--preorder")==0)
            {
                bfa.preorder(&bfa.gyoker);
            }
            else if(strcmp(argv[1],"--inorder")==0)
            {
                bfa.inorder(&bfa.gyoker);
            }
            else if(strcmp(argv[1],"--postorder")==0)
            {
                bfa.postorder(&bfa.gyoker);
            }
            else
            {
                usage();
            }
        }
        else
        {
            usage();
        }
        bfa.destroy_tree(&bfa.gyoker);
        return 0;
    }
```

Tanulságok, tapasztalatok, magyarázat:

A programot átírtuk "C"-ből "C++"-ba. Annyival egészítettük ki az eredeti programot, hogy létrehoztunk egy binfa nevű osztályt(class Binfa), és ebben még egy "public" és egy "private" részt. Még annyi változás történt, hogy létrehoztuk a binfa objektumot a "main"-en belül, és azon keresztül lehet elérni a binfa adatait, nem "insertet" kell meghívni, hanem a "balra bitshift" operátort.

6.5. Mutató a gyökér

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

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/5.cpp

```
    #include <iostream>
    #include <cstdlib>
    #include <ctime>
    #include <string.h>

    #define null NULL

    class Binfa
```

```
{
private:
    class Node
    {
    public:
        Node(char c='/')
        {
            this->c=c;
            this->left = null;
            this->right = null;
        }
        char c;
        Node* left;
        Node* right;
    };
    Node* fa;


public:
    Binfa()
    {
        gyoker=fa=new Node();
    }


    void operator<<(char c)
    {
        if(c=='0')
        {
            if(fa->left == null)
            {
                fa->left = new Node('0');
                fa = gyoker;
            }
            else
            {
                fa = fa->left;
            }
        }
        else
        {
            if(fa->right == null)
            {
                fa->right = new Node('1');
                fa = gyoker;
            }
            else
            {
                fa = fa->right;
            }
```

```
        }
    }

    void preorder(Node* elem,int depth=0)
    {
        if(elem==null)
        {
            return;
        }
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
                spaces[i+1]='-';
            }
            spaces[depth*2]='\0';

            printf("%s%c\n",spaces,elem->c);
        }
        else
        {
            printf("%c\n",elem->c);
        }
        preorder(elem->left,depth+1);
        preorder(elem->right,depth+1);
    }

    void inorder(Node* elem,int depth=0)
    {
        if(elem==null)
        {
            return;
        }
        inorder(elem->left,depth+1);
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
                spaces[i+1]='-';
            }
            spaces[depth*2]='\0';

            printf("%s%c\n",spaces,elem->c);
        }
```

```
            else
            {
                printf("%c\n",elem->c);
            }
            inorder(elem->right,depth+1);
        }

        void postorder(Node* elem,int depth=0)
        {
            if(elem==null)
            {
                return;
            }
            postorder(elem->left,depth+1);
            postorder(elem->right,depth+1);
            if(depth)
            {
                char *spaces;
                spaces =(char*) malloc(sizeof(char)*depth*2+1);
                for(int i=0;i<depth;i+=2)
                {
                    spaces[i]='-';
                    spaces[i+1]='-';
                }
                spaces[depth*2]='\0';

                printf("%s%c\n",spaces,elem->c);
            }
            else
            {
                printf("%c\n",elem->c);
            }
        }

        void destroy_tree(Node* elem)
        {
            if(elem==null)
            {
                return;
            }
            destroy_tree(elem->left);
            destroy_tree(elem->right);
            if(elem->c!='/') delete elem;
        }

        Node* gyoker;

    };

    void usage()
```

```
{
    printf("Használat: ./binfa KAPCSOLÓ\n");
    printf("Az KAPCSOLÓ lehet:\n");
    printf("--preorder\tA bináris fa preorder bejárása\n");
    printf("--inorder\tA bináris fa inorder bejárása\n");
    printf("--postorder\tA bináris fa postorder bejárása\n");
}

int main(int argc, char** argv)
{
    srand(time(0));
    Binfa bfa;
    for(int i=0;i<100;i++)
    {
        int x=rand()%2;
        if(x)
        {
            bfa<<'1';
        }
        else
        {
            bfa<<'0';
        }
    }
    if(argc == 2)
    {
        if(strcmp(argv[1],"--preorder")==0)
        {
            bfa.preorder(bfa.gyoker);
        }
        else if(strcmp(argv[1],"--inorder")==0)
        {
            bfa.inorder(bfa.gyoker);
        }
        else if(strcmp(argv[1],"--postorder")==0)
        {
            bfa.postorder(bfa.gyoker);
        }
        else
        {
            usage();
        }
    }
    else
    {
        usage();
    }
    bfa.destroy_tree(bfa.gyoker);
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat:

A program az előző feladat alapja, csak ebben a feladatban módosítottuk. A "public" részen belül a "Binfa" nevű függvényt változtattuk meg első sorban. tehát itt a gyökér is egy pointer legyen, mint a fa. Az előzben a gyökér egy objektum volt, aminek terület is volt foglalva. Itt pedig a gyökér egy pointer, aminek kell adni egy értéket, hogy hova mutasson. Itt nincsen referencia gyökér, csak sima gyökér van(ez azt jelenti, hogy ahol eddig ki volt téve az "and" jel, ott nem teszük ki a "bfa" előtt).

6.6. Mozgató szemantika

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

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/6.cpp

A feladat megoldásában segített: Lovász Botond.

Tanulságok, tapasztalatok, magyarázat:

A program alapja az előző program, a működése is hasonló. Annyiban különbözik, hogy itt beleépítettünk egy másoló konstruktort, ez nagy szerepet játszik az argumentum átadásban és a visszatérési érték használatában. A fordító automatikusan létrehoz egy másoló konstruktort, hacsak nem definiálunk egyet.

A feldat megoldásához használnunk kell a konstruktort, a másolót és a destruktort. A bináris fák mozgatásához másolásához, ismétlődő lépésekből álló műveletsorozaton alapuló rekurzív függvényeket használunk. A destruktor alapvetően üres, ezt ki kell fejteni, mert e nélkül a program nem fog lefordulni. A másoló konstruktor, és az "=" operátor nincs értelmezve, mert általában csak az objektumokra mutató referenciát küldjük ide-oda. Destruktor: az objektum törlődése előtti teendőket hajtja végre(erőforrások felszabadítása). Másoló konstruktor: lefoglaljuk az objektum üzemeltetéséhez szükséges erőforrásokat, majd egy másik példány másolataként hozzuk létre az új objektumot.

DRAFT

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

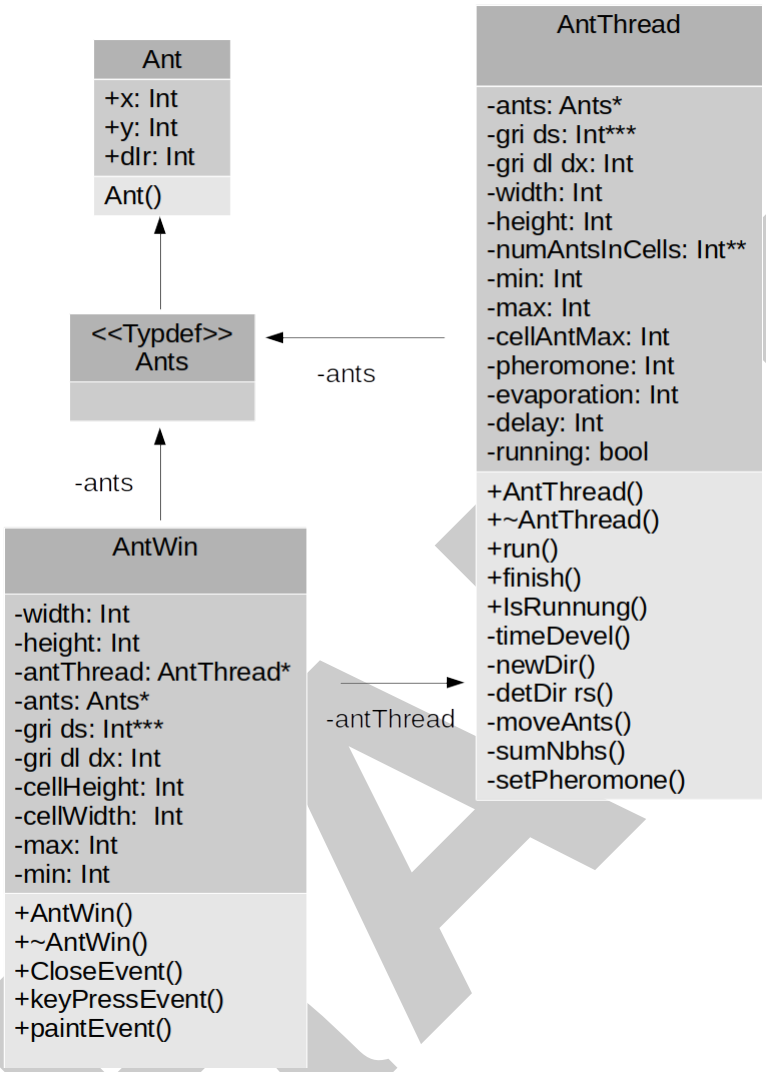
Írj Qt C++-ban egy hangyaszimulációs programot, a forrásaidról utólag reverse engineering jelleggel készíts UML osztálydiagramot is!

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Myrmecologist?fbclid=IwAR28U
jJNiCGYkG-fYlLgwqk-YyOyErWjrgrpJPg

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

A feladat megoldásában segítettem: Boros István.

DRAFT



A Diagramm magyarázata:

Ha "+" jel van egy tag előtt, akkor az publikus, tehát látható egy másik osztályból is, ha "-" jel van akkor nem látható.

Négy osztályból áll a kód: Ant, Typdef, AntWin, AntThread. Az "Ant" osztályban van a hangya tulajdonságai. Az "x" jelöli az sort, az "y" pedig az oszlopot, ebből a két számból megtuduk a hangya pontos helyét. Ebben a részben még megadjuk a hangya irányát is(dir).

Az "AntWin" osztályban található a felugró ablak szélessége(width), és magassága(height). Ebben az osztályba meghívjuk a "AntThread" osztályt, és más pointerek által jelölt kódrészeket. Itt adjuk meg a cellák méretét(cellHeight, cellWidth). Valamint a bllentyűzet eseményeket(KeyPressEvent) is itt dolgozzuk fel. Az ablak bezárását(CloseEvent) is itt írjuk bele a prgramba, és magát a rajzolást(paintEvent) is. A "+AntWin()" függvény egy konstruktor és az építi fel az objektumot, a "+~AntWin()" függvény pedig a destruktor.

A "Typdef" rész, definiálja az "Ants" típust.

Az "AntThread" osztály meg van hívva az AntWin osztályban. Itt adjuk meg a hangyáknak a követési útvonalat. Ebben a részben megszámoljuk a maximális hangyaszámot egy cellában(cellAntMax) és megadjuk a feromont(ezt követik a hangyák(pheromone)). Ebben a részben mozgatjuk(moveAnts) a hangyákat a feromon(setPheromone) után.

DRAFT

7.2. Java életjáték

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

Megoldás forrása: https://github.com/Matchbox1233/Conway/blob/master/2.java

Tanulságok, tapasztalatok, magyarázat:

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása: https://github.com/Matchbox1233/Conway/blob/master/3.cpp

Tanulságok, tapasztalatok, magyarázat:

A programot az alábbi paranccsal lehet lefordítani: "g++ 3.cpp -o sfml-app -lsfml-graphics -lsfml-window -lsfml-system". Megnyitni a szokásos módon lehet. Amikor a program elindult, akkor kézzel kell rajzolnunk egy alakzatot, majd nyomni egy "e" betűt, hogy a program elkezdje az "életjátékot"

Conway-féle életjáték kis háttértörténete: Az életjátékot John Conway találta ki, aki Cambridge Egyetem matematikus. Ez egy nullszemélyes játék, annyi a szerepe, hogy az ember megad egy kezdőalakot, és azután elindul a játék. Ez a játék matematikailag a sejtautómaták közé tartozik.

A program elég sok osztályból áll, és persze egy fő részből(main). Az első osztály a "Grid" nevet kapta, ebben található egy "public" rész, ami meghatározza a program kezdetén lévő ablak nagyságát. Aztán a "public" rész után jön a "void draw", ez a "void" felelős az ablak megrajzolásáért, ami megjelenik. Ez az megjelenített ablak négyzetrácsos lesz. A második osztály a "Square" nevet kapta. Ez a rész felelős azért, hogy ha rákattintunk egy négyzetre a rácshálón, akkor kitöltse fekete színnel. A harmadik rész az egy vektor lesz, ezzel a vektorral mozgatjuk a kijelölt négyzeteket a játékszabálynak megfelelően, ami a következő: Egy sejttel egy körben három dolog történhet. Az első, ha a sejt túléli a kört(ha két vagy három szomszédja van). A második, ha a sejt elpusztul a körben(ha kettőnél kevesebb, vagy háromnál több szomszédja van). A harmadik pedig, ha egy új sejt születik(ha a környezetében pontosan három sejt található). Bővebb információ az életjáték szabályairól: https://hu.wikipedia.org/wiki/%C3%89letj%C3%A1t%C3%A9k A negyedik rész a "void killall", itt felszabadítjuk az előző részben(vector) használt pointereket, így nem terhelünk felesleges memóriát a továbbiakban. Az ötödik rész a fő rész a "main". Ebben a részben megadjuk a megnyíló ablak paramétereit. Aztán a "push_back" -el hozáadunk egy elemet a "vector" -hoz. A "while" függvény addig megy, amíg az ablak nyitva van. Először "letisztítja" a képernyőt. Ha nem kattintunk egy négyzetre sem, akkor bezárdóik az ablak, ha viszont kattintunk a bal egérgombbal, akkor kiszinezi a négyzetet. Ezen kívül, ha megnyomjuk a "Q"-t akkor bezáródik a program, ha a "C"-t nyomjuk meg akkor "megöli" az összes négyzetet, és ha "E"-t nyomunk, akkor meg szerkesztő módba lépünk át. Végül meghívjuk a "draw" osztályt a "main"-be, és "update"-jük a négyzeteket.

7.4. BrainB Benchmark

Megoldás forrása: https://github.com/nbatfai/esport-talent-search?fbclid=IwAR0n58sBL-LrZ5N9rpC-zftv0CeiW

Tanulságok, tapasztalatok, magyarázat:

DRAFT

A kód elég öszetett, hiszen nyolc fájlól áll. Ezek között van egy "használíti utasítás" ez a "README.md", valamint egy licensz információ(LICENSE). A program fő részre a "main.cpp"-ben található, és ehhez kapcsolódik a több kódrészlet, amik külön állományokban vannak elhelyezeve a jobb átláthatóság érdekében.

A program elndításához, és futatásához érdemes elolvasni a "README" szöveges dokumentumot. A program célja, hogy fejlessze a koncentrációs és a követési képességeinket. Amikor elindítjuk ezt a programot először megjelenik négy darab négyzet, és mindegyiknek a közepén van egy kék kör. Ezen a kék körön kell tartanunk az egeret lenyomva(bal egér gomb). A négyzet mozogni fog, ha sikeresen rajta tartjuk az egeret a kék körön, akkor újabb négyzetek jelennek meg, ha viszont nem sikerül, akkor pedig csökken a négyzetek száma. A követés egyre nehezbb lesz, mert a sok négyzet egyre gyakrabban, fogja egymás fedni, és akkor már elég nehéz nyomon követni a mi kis kék körünket.

DRAFT

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

Megoldás forrása: https://github.com/Matchbox1233/Schwarzenegger/blob/master/1.py

```
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist. ↩
   load_data()

x_train.shape

# Reshaping the array to 4-dims so that it can work with the Keras API
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
# Making sure that the values are float so that we can get decimal  ↩
   points after division
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# Normalizing the RGB codes by dividing it to the max RGB value.
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train', x_train.shape[0])
print('Number of images in x_test', x_test.shape[0])

# Importing the required Keras modules containing model and layers
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
```

```
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten()) # Flattening the 2D arrays for fully connected ↩
       layers
    model.add(Dense(128, activation=tf.nn.relu))
    model.add(Dropout(0.2))
    model.add(Dense(10,activation=tf.nn.softmax))

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    model.fit(x=x_train,y=y_train, epochs=10)

    model.evaluate(x_test, y_test)

    image_index = 4444
    plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
    pred = model.predict(x_test[image_index].reshape(1, 28, 28, 1))
    print(pred.argmax())
    print(y_test[image_index])
```

Tanulságok, tapasztalatok, magyarázat:

A program lényege, hogy kézzel rajzolt számokat ismerjen fel. A programben először importáljuk a "tensorflow"-t(import tensorflow as tf). A "tensorflow" segítségével töltjük be az MNIST adatbázisban tárolt adatokat. Az MNIST adatbázisban található képeket, a hozzá tartozó címkéket, illetve a tesztképeket és címkéket eltároljuk az "x_train", "y_train", "x_test", "y_test" változókban(x_train = x_train.reshape), (x_test = x_test.reshape). A következőkben az "x_train" és "x_test" értékeit konvertáljuk "float" típusúra(x_train = x_train.astype('float32'). Ezek után normalizáljuk az RGB értékeket. A neurális háló felépítéséhez szükséges hat modult hozzáadnunk a programhoz(model.add...). A hozzáadás után létrehozzuk a hálót, és fel is tanítjuk. Végül pedig megnézzük a háló tudását, ebben segítségünkre vannak a tesztképek és címkék.

8.2. Mély MNIST

Python

Megoldás forrása: https://github.com/Matchbox1233/Schwarzenegger/blob/master/2.py

A feladat megolsásában segített: Petrus József Tamás

```
    from __future__ import print_function

    from tensorflow.examples.tutorials.mnist import input_data
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

    import tensorflow as tf

    def weight_variable(shape):
```

DRAFT

```
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')

# Input layer
x  = tf.placeholder(tf.float32, [None, 784], name='x')
y_ = tf.placeholder(tf.float32, [None, 10],  name='y_')
x_image = tf.reshape(x, [-1, 28, 28, 1])

# Convolutional layer 1
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

# Convolutional layer 2
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

# Fully connected layer 1
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])

W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# Dropout
keep_prob  = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# Fully connected layer 2 (Output layer)
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2, name='y')
```

```
# Evaluation functions
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),  ↩
   reduction_indices=[1]))

correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32), name ↩
   ='accuracy')

# Training algorithm
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

# Training steps
with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())

  max_steps = 1000
  for step in range(max_steps):
    batch_xs, batch_ys = mnist.train.next_batch(50)
    if (step % 100) == 0:
      print(step, sess.run(accuracy, feed_dict={x: mnist.test.images,  ↩
         y_: mnist.test.labels, keep_prob: 1.0}))
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys,  ↩
       keep_prob: 0.5})
  print(max_steps, sess.run(accuracy, feed_dict={x: mnist.test.images,  ↩
     y_: mnist.test.labels, keep_prob: 1.0}))
```

Tanulságok, tapasztalatok, magyarázat:

A program elején importáljuk a szükséges modulokat(pl:tensorflow). Utána definiálunk négy függvényt(ezeket a "def" jelzi), ezek a függvények a program rövidítésében segítenek. A függvény definiálások után létrehozzuk a neurális háló rétegeit(ezt jelzi a "layer" szó), ebben a kódban öt rétegű a neurális háló. A hálók megadása után meghatározzuk, hogy milyen módon akarjuk kiértékelni a bemenetet. Majd megadjuk, hogy milyen algoritmussal tanítjuk a hálót(Training algorithm, Training steps). A kód végére pedog megadjuk, hogy minden századik iterációban leteszteljük a háló tudását.

8.3. Minecraft-MALMÖ

A feladdat megoldásában segített: Petrus József Tamás

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

Megoldás forrása: https://github.com/Matchbox1233/Schwarzenegger/blob/master/3.py

Tanulságok, tapasztalatok, magyarázat:

A program célja, hogy Steve ne akadjon el a pályán öt percen keresztül. Steve az előtte lévő 12 blokkot vizsgálja. Adottak feltételek, ha ezek megfelelnek, akkor ugrik, vagy irányt vált egy másik irányba. Stevenek még sokat kell fejlődnie, mert a láva tavakat és a szakadékokat még nem tudja kikerülni.

A programba először importáljuk a szükséges modulokat, illetve a "MALMÖ API"-t. Az importálások után definiálunk pár függvényt. A "restart_minecraft" föggvény feladata, hogy a mission lefutása után a prgram tudja fogadni a következő missiont. A "run" függvényben adjuk meg a pontos missiont illetve az ez alatti vezérést. A "missionXML"-ben van megadva Steve kiindulási helyzete, a világ generálásának szabályai és, hogy figyelje a körülötte klévő dolgokat. Ezután megadjuk a mission hosszát időben(my_mission.timeLimitInSeconds(300)), ez azt jelenti, hogy háromszáz másodpercig fog futni, valamint megadjuk a Minecraft felbontását is, ami 640x480-as lesz(my_mission.requestVideo(640, 480)). A következőkben megadjuk, hogy a Minecraft kliens milyen porton és címen érhető el, hogy a program tudjon csatlakozni. Következik két "while" ciklus. Az elsőben a ciklus addig vár, amíg a kliens készen áll a program általi irányítást fogadni. A másodikban van Steve irányítása, ez több elágazásból áll. Az első kettőben megadjuk, hogy Steve fel tudjon vagy át tudjon ugrani egy akadályt, és ne egyfolytában ugráljon. A harmadik elágazásban van a jobbra, ballra fordulás. A negyedik elágazásban Steve eldönti, hogy melyik égtáj felé nézzen. A "bloccks" nevű kétdimenziós tömbbe töltjük a megvizsgált blokkok nevét. Az alapján, hogy Steve melyik égtáj felé néz megvizsgálja az előtte lévő blokkokat, ez alapján dönt, hogy ugorjon vagy forduljon.

DRAFT

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása: https://github.com/Matchbox1233/Chaitin/blob/master/1a.lisp

A feladat megoldásában segítettem: Boros István.

```
Iteratív faktoriális számolás Lisp-ben:

    (defun fact(x)
    (setq sum 1)
    (loop while (> x 1)
        do (
            setq sum (* sum x)
            x (- x 1)
            )
        )
        sum
    )

    (setq f (read))

    (format t "~D! ~D" f (fact f))
```

Tanulságok, tapasztalatok, magyarázat:

Először is definiálunk egy "fact" nevű függvényt(defun fact(x), ez fog felelni a faktoriális számolásáért. A függvény megadása után bekérünk egy számot a felhsználótól(setq sum 1). Ez lesz az a szám, aminek meg akarjuk határozni a faktoriálisát. A "do-while" ciklusban található, a faktoriális számolás matematikai műveletei. Végül a számra meghívjuk a "fact" függvényt.

Megoldás forrása: https://github.com/Matchbox1233/Chaitin/blob/master/1b.lisp

```
Rekurrzív faktoriális számolás Lisp-ben:
```

```
(defun factorial(x)
    (if (= x 1)
        (setq a 1)
    )
    (if (> x 1)
        (setq a (* x (factorial (- x 1))))
    )

    a
)

(format t "~D! is ~D" 5 (factorial 5))
```

Tanulságok, tapasztalatok, magyarázat:

A programban először definiáljuk a "factorial" nevű függvényt(defun factorial(x)). Ez a függvény fogja rekurzív módon kiszámolni a faktoriális értékét. A két "if"-ben a matematikai műveleteket láthatjuk. A program végén pedig meghívjuk a "factorial" függvényt a ötös számra.

9.2. Gimp Scheme Script-fu: króm effekt

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

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

Megoldás forrása: https://github.com/Matchbox1233/Chaitin/blob/master/2.scm

```
(define (color-curve)
(let* (
    (tomb (cons-array 8 'byte))
)
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
tomb)
)

;(color-curve)

(define (elem x lista)

    (if (= x 1) (car lista) (elem (- x 1) ( cdr lista ) ) )
```

```
)

(define (text-wh text font fontsize)
(let*
    (
        (text-width 1)
        (text-height 1)
    )

    (set! text-width (car (gimp-text-get-extents-fontname text fontsize ↩
        PIXELS font)))
    (set! text-height (elem 2  (gimp-text-get-extents-fontname text  ↩
       fontsize PIXELS font)))

    (list text-width text-height)
    )
)

;(text-width "alma" "Sans" 100)

(define (script-fu-bhax-chrome text font fontsize width height color  ↩
   gradient)
(let*
    (
        (image (car (gimp-image-new width height 0)))
        (layer (car (gimp-layer-new image width height RGB-IMAGE "bg"  ↩
           100  LAYER-MODE-NORMAL-LEGACY)))
        (textfs)
        (text-width (car (text-wh text font fontsize)))
        (text-height (elem 2 (text-wh text font fontsize)))
        (layer2)
    )

    ;step 1
    (gimp-image-insert-layer image layer 0 0)
    (gimp-context-set-foreground '(0 0 0)) // a ' miatt nem lesz  ↩
       függvény, szabadnak tekinti a lips
    (gimp-drawable-fill layer  FILL-FOREGROUND )
    (gimp-context-set-foreground '(255 255 255))

    (set! textfs (car (gimp-text-layer-new image text font fontsize  ↩
       PIXELS)))
    (gimp-image-insert-layer image textfs 0 0)
    (gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (-  ↩
       (/ height 2) (/ text-height 2)))

    (set! layer (car(gimp-image-merge-down image textfs CLIP-TO-BOTTOM- ↩
       LAYER)))
```

```
        ;step 2
        (plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)

        ;step 3
        (gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE ↩
           )

        ;step 4
        (plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

        ;step 5
        (gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
        (gimp-selection-invert image)

        ;step 6
        (set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2"  ↩
           100 LAYER-MODE-NORMAL-LEGACY)))
        (gimp-image-insert-layer image layer2 0 0)

        ;step 7
        (gimp-context-set-gradient gradient)
        (gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY  ↩
           GRADIENT-LINEAR 100 0 REPEAT-NONE
            FALSE TRUE 5 .1 TRUE width (/ height 3) width  (- height (/  ↩
               height 3)))

        ;step 8
        (plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5  ↩
           5 0 0 TRUE FALSE 2)

        ;step 9
        (gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

        (gimp-display-new image)
        (gimp-image-clean-all image)
        )
    )

    ;(script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 '(255 0 0)  ↩
       "Crown molding")

    (script-fu-register "script-fu-bhax-chrome"
        "Chrome3"
        "Creates a chrome effect on a given text."
        "Norbert Bátfai"
        "Copyright 2019, Norbert Bátfai"
        "January 19, 2019"
        ""
        SF-STRING       "Text"      "Bátf41 Haxor"
        SF-FONT         "Font"      "Sans"
```

```
        SF-ADJUSTMENT   "Font size" '(100 1 1000 1 10 0 1)
        SF-VALUE        "Width"     "1000"
        SF-VALUE        "Height"    "1000"
        SF-COLOR        "Color"     '(255 0 0)
        SF-GRADIENT     "Gradient"  "Crown molding"
    )
    (script-fu-menu-register "script-fu-bhax-chrome"
        "<Image>/File/Create/BHAX"
    )
```

Tanulságok, tapasztalatok, magyarázat:

A program lényege:

A program első részében meghatározzuk(define) a színeket, a méretet, a betű típust, a betűméretet. Az utolsó "define"-ban létrehozunk egy "image" és egy "layer" nevű változót, amikben eltároljuk a létrehozott kép adatait(szín, szélesség, magasság), itt még megadjuk a szöveg méretét is. Utána jönnek a lepések(step), hogy hogyan is kell megcsinállni Lisp-ben a Chrome effektet

Első lépés: Megadunk egy fehér hátteret(gimp-context-set-foreground '(0 0 0)), majd azt kitöltjük(FILL-FOREGROUND) fekete színnel(gimp-context-set-foreground '(255 255 255)). Majd erre a háttére írunk egy kis szöveget fehér színnel(set! textfs), amit középre igazítunk(offsets textfs). A lépés végén pedig összeillesztjük a képet, és a rétegünket(set! layer)

A második lépésben csinálunk egy Gauss(gauss-iir) elmosást a megalkotott képünkre.

A harmadik lépésben beállítjuk a színek alsó és felső határait. Ez élesíteni fog a képen.

Negyedik lépésként ismét egy Gauss elmosás következik(gauss-iir).

Az ötödik lépésben a színszerinti kijelölést(select-color) alkalmazva kijelöljük a fekete részt, és invertáljuk(selectio invert). Így a végén a felirat körl kapunk egy vékony keretet.

A hatodik lépésben létrehozunk egy új áttetsző réteget(layer-new image). Ezt a réteget állítjuk be fő rétegként a képre.

A hetedik lépésben meghatározunk egy "gradient"(et-gradient gradient), ebben az esetben, ez egy szürke színátmenet. A hetedik lépés utolsó sora felel azért, hogy kattintásssal is meg tudjuk csinálni a színátmenetet.

A nyolcadik lépésben megadjuk a kép térhatását(RUN-NONINTERACTIVE). Ennek vannak különböző paraméterei(layer 120 25 7 5 5 0 0).

A kilencedik lépésnél egy görbe segítségével tudjuk beállítani a fémességét(curves-spline). Ebben a lépsében meghívjuk a kód elején meghatározott színeket, ezt jelöli a ""color-curve" függvény. A "define"-ban meghatároztuk a fémesség színét, ezt jelzik a számok.

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

DRAFT

```
(define (elem x lista)

(if (= x 1) (car lista) (elem (- x 1) ( cdr lista ) ) )

)

(define (text-width text font fontsize)
(let*
    (
        (text-width 1)
    )
    (set! text-width (car (gimp-text-get-extents-fontname text fontsize ↩
        PIXELS font)))

    text-width
    )
)

(define (text-wh text font fontsize)
(let*
    (
        (text-width 1)
        (text-height 1)
    )
    ;;;
    (set! text-width (car (gimp-text-get-extents-fontname text fontsize ↩
        PIXELS font)))
    ;;; ved ki a lista 2. elemét
    (set! text-height (elem 2  (gimp-text-get-extents-fontname text  ↩
       fontsize PIXELS font)))
    ;;;

    (list text-width text-height)
    )
)


;(text-width "alma" "Sans" 100)

(define (script-fu-bhax-mandala text text2 font fontsize width height  ↩
   color gradient)
(let*
    (
        (image (car (gimp-image-new width height 0)))
        (layer (car (gimp-layer-new image width height RGB-IMAGE "bg"  ↩
           100 LAYER-MODE-NORMAL-LEGACY)))
        (textfs)
        (text-layer)
```

```
            (text-width (text-width text font fontsize))
            ;;;
            (text2-width (car (text-wh text2 font fontsize)))
            (text2-height (elem 2 (text-wh text2 font fontsize)))
            ;;;
            (textfs-width)
            (textfs-height)
            (gradient-layer)
        )

        (gimp-image-insert-layer image layer 0 0)

        (gimp-context-set-foreground '(0 255 0))
        (gimp-drawable-fill layer FILL-FOREGROUND)
        (gimp-image-undo-disable image)

        (gimp-context-set-foreground color)

        (set! textfs (car (gimp-text-layer-new image text font fontsize  ↩
           PIXELS)))
        (gimp-image-insert-layer image textfs 0 -1)
        (gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2))  (/ ↩
            height 2))
        (gimp-layer-resize-to-image-size textfs)

        (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
        (gimp-image-insert-layer image text-layer 0 -1)
        (gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
        (set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO- ↩
           BOTTOM-LAYER)))

        (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
        (gimp-image-insert-layer image text-layer 0 -1)
        (gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
        (set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO- ↩
           BOTTOM-LAYER)))

        (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
        (gimp-image-insert-layer image text-layer 0 -1)
        (gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
        (set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO- ↩
           BOTTOM-LAYER)))

        (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
        (gimp-image-insert-layer image text-layer 0 -1)
        (gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
        (set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO- ↩
           BOTTOM-LAYER)))

        (plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
```

```
        (set! textfs-width (+ (car(gimp-drawable-width textfs)) 100))
        (set! textfs-height (+ (car(gimp-drawable-height textfs)) 100))

        (gimp-layer-resize-to-image-size textfs)

        (gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width  ↩
           2) (/ textfs-width 2)) 18)
            (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) ↩
                (+ textfs-height 36))
        (plug-in-sel2path RUN-NONINTERACTIVE image textfs)

        (gimp-context-set-brush-size 22)
        (gimp-edit-stroke textfs)

        (set! textfs-width (- textfs-width 70))
        (set! textfs-height (- textfs-height 70))

        (gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width  ↩
           2) (/ textfs-width 2)) 18)
            (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) ↩
                (+ textfs-height 36))
        (plug-in-sel2path RUN-NONINTERACTIVE image textfs)

        (gimp-context-set-brush-size 8)
        (gimp-edit-stroke textfs)

        (set! gradient-layer (car (gimp-layer-new image width height RGB- ↩
           IMAGE "gradient" 100 LAYER-MODE-NORMAL-LEGACY)))

        (gimp-image-insert-layer image gradient-layer 0 -1)
        (gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
        (gimp-context-set-gradient gradient)
        (gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL- ↩
           LEGACY GRADIENT-RADIAL 100 0
        REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+  ↩
           (+ (/ width 2) (/ textfs-width 2)) 8) (/ height 2))

        (plug-in-sel2path RUN-NONINTERACTIVE image textfs)

        (set! textfs (car (gimp-text-layer-new image text2 font fontsize  ↩
           PIXELS)))
        (gimp-image-insert-layer image textfs 0 -1)
        (gimp-message (number->string text2-height))
        (gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- ↩
            (/ height 2) (/ text2-height 2)))

        ;(gimp-selection-none image)
        ;(gimp-image-flatten image)

        (gimp-display-new image)
```

```
        (gimp-image-clean-all image)
        )
    )

    ;(script-fu-bhax-mandala "Bátfai Norbert" "BHAX" "Ruge Boogie" 120 1920 ↩
        1080 '(255 0 0) "Shadows 3")

    (script-fu-register "script-fu-bhax-mandala"
        "Mandala9"
        "Creates a mandala from a text box."
        "Norbert Bátfai"
        "Copyright 2019, Norbert Bátfai"
        "January 9, 2019"
        ""
        SF-STRING       "Text"      "Bátf41 Haxor"
        SF-STRING       "Text2"     "BHAX"
        SF-FONT         "Font"      "Sans"
        SF-ADJUSTMENT   "Font size" '(100 1 1000 1 10 0 1)
        SF-VALUE        "Width"     "1000"
        SF-VALUE        "Height"    "1000"
        SF-COLOR        "Color"     '(255 0 0)
        SF-GRADIENT     "Gradient"  "Deep Sea"
    )
    (script-fu-menu-register "script-fu-bhax-mandala"
        "<Image>/File/Create/BHAX"
```

Tanulságok, tapasztalatok, magyarázat:

A program lényege, hogy egy Mandalát rajzooljunk Gimp-ben, de úgy, hogy ne grafikusan, hanem csak kóddal.

A program elején definiálunk három függvényt, hogy ezekre később tudjunk hivatkozni. Ezek segítik rövidíteni a programot. Az első defineban megadjuk, hogy a "car" függvény ne csak az első elemet választja ki. A második definiálásban megadjuk a szöveg szélességét(text-width 1) és magasságát(text-height 1). A haramdik definiálásban(define (script-fu-....) létrehozunk egy új képet, és ennek megadhatjuk a szélességét, a magasságát, és a színét(ezt a nulla jelzi)(gimp-image-new width height 0). A föggvény definiálás után megadjuk az előtérszínt(set-foreground), majd ezzel a színnel kitöltjük az előteret(gimp-drawable-fill). Az első "set!"-ben megadjuk, kiíratjuk magát a szöveget. A szöveg szélességét(/ width 2), és magasságát(/ height 2) is megadjuk oópixelben(gimp-text-layer-new image text font fontsize PIXELS). Illetve még megadjuk a szöveg magasságát, szélességét és a rajta lévő szöveg szélességét. A következő négy "step!" forgatások, itt állítjuk be, hogy a szöveg mmilyen módon legyen elforgatva, ezt jelzi a "ROTATE"-t szó. Az eslőben 180 fokkal(ROTATE-180), a másodikban pi/2 -vel(rotate text-layer (/ *pi* 2)), a harmadikban pi/4- el(rotate text-layer (/ *pi* 4)), a negyedíikben pedig pi/6 -al foragtunk(rotate text-layer (/ *pi* 6)). A forgatások után méreteket, középre igazítások, valamit egy fajta "stílust" a színeknek, ez lehet mondjuk egy színátmenet is. Ezek után megadunk egy új réteget(new image text2), amin létrehozunk egy új szöveget, ezt is középre igazítjuk. A program végén, még állíthajuk, hogy a kép az elkészítés után ki legyen jelölve, vagy ne. Ha azt akarjuk, hogy ki legyen jelölve, akkor egy ";"-t kell írnunk a sor elé: (gimp-selection-none image), ha viszont nem, akkor csak kitöröljük a ";"-t.

10. fejezet

Helló, Olvasónapló! (Gutenberg)

10.1. Magas szintű programozási nyelvek 1 by Juhász István (Pici könyv)

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

A programozási nyelveknek három szintje van, ezek a gépi nyelv, az assembly szintű nyelv, a magas szintű nyelv. A magas szintű nyelven megírt programot forrásprogramnak vagy forrászövegnek nevezzük. A forrásokra mind nyelvtanilag és mind formailag is vonatkoznak szabályok ezeket a szabályokat szintaktikai szabályoknak hívjuk. Minden processzor saját gép nyelvvel rendelekzik, tehát a forrsszövegből gépi nyelvet kell csinálnunk. Erre kéféle módszer létezik, a fordítóprogramos és az interpreteres. A fordítóprogram a következőket hajtja végre: lexikális elemzés, szintaktikai elemzés, szemantikai elemzés, kódgenerálás. A lexikális elemző feldarabolja lexikális egységekre a forrásszöveget. A szintaktikai elemzés ellenőrzi, hogy teljesülnek e a szintaktikai szabályok. Futattható programot a szerkesztő készíti el. A már futattható programot a betöltő tölti be. A futó programot a futattó rendszer felügyeli. A fordító programok tetszőleges nyelvről tudnak tetszőleges nyelvre fordítani. Az interpreteres technika esetenén is megvan az első három lépés, de itt nem készül tárgyporgram, rögtön kapjuk az eredményt. A programnyelvek vagy fordítóprogramosak, vagy interpreteresek, vagy együttesen alkalmazzák mindkettőt. Minden programnyelvnek megvan a saját szabványa, ezt hivatkozási nyelvnek hívjuk. Ebben definiálva vannak a szintaktikai és szemantikai szabályok általában angolul. Léteznek még az implementációk is, ezek nem kompatibilisek egymással és a hivatkozási nyelvvel sem. A mai napig probléma a hordozhatóság(egyik implementációból viszek a kódot egy másik implementációba). Napjainkban a programíráshoz már grafikus felületet használunk, ami tratalmaz szövegszerkesztőt, fordítót, kapcsolatszerkesztőt, betöltőt, futattó rendszert és belövőt

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

Az adatabsztrakció elsı megjelenési formája az adattípus. az adattípusnak van neve, ami egy azonosító. Valamely programozási nyelvek ismerik ezt az aeszközt, valamelyek nem, ennek megfelelően beszélünk típusos és nem típusos nyelvekről. Az adattípust meghatározza: a tartomány, a műveletek, a reprezentáció. Minden adattípus mögött van egy megfelelı belsı ábrázolási mód. Valamely programozási nyelvek megengedik, hogy a programozó definiálhasson típusokat. Az adattípusoknak két nagy csoportja van a skalár vagy egyszerű, és a strukturált vagy összetett. Az egyszerű adattípus elemei atomiak(nem bontható tovább), literálként megjelenhetnek. Az öszetett adattípus elemei pedig egy-egy értékcsoport(nem atomi), literálként nem jelenhetnek meg.

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

DRAFT

Egy programozási eszköz, aminek három komponense van: a név, a típus, és az érték. Mindig deklarálni kell. Mindig névvel jelenik meg, és ez mindig értékkomponenst jelent. Szerepe egyrészt, hogy a sokszer előforduló értékeknek "beszélő" neveket adjunk. Másrészt, hogy ha át akarunk írni egy értéket, akkor nem kell az egész programon keresztül ezt megtennünk, elég ha csak a deklarációs utasításban átírjuk.

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 lehetőségünk is van: az értékadó utasítás, a 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, a "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, a "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.

VI. heti előadás (72. oldal, a "A program szerkezete" című rész):

Az eljárásorinetált programnyelvekben a program szövege többé-kevésbé független, programegységekre tagolható. Négy fajta programozásegység létezik az eljárás orientált nyelvekben: alprogram, blokk, csomag, taszk. Az alprogram egy absztrakciós eszköz, amely egy bemeneti adatcsoportot képez le egy kimeneti adatcsoportra. Ismerjük a specifikációt, de nem ismerjük az implementációt. Az alprogramot többször is fel tudjuk használni, ha a program különböző pontjain ugyanaz a programrész megismétlődik. Az ismétlédő programrészt elég egyszer megírnunk, és később csak hivatkoznunk kell rá. Formálisan az alprogram fejből vagy specifikációból, törzsből vagy implemetációból, és végből épül fel. Négy komponensből áll: név, formális paraméter lsita, törzs, környezet. Az alprogramban találhatók lokális, és globálsi nevek is. A lokális nevek az aplrogramon kívülről nem láthatók. A globális neveket nem az adott aplrogramban deklarálunk, hanem rajta kívül, viszont a törzsben szabályosan hivatkozunk rájuk. Egy aplrogram környezet

DRAFT

alatt a globális változók együttesét értjük. Az alprogramoknak két fajtája van: eljárás és függvény. Az eljárás valamilyen tevékenységet hajt végre. A függvény feldata, hogy egyetlen értéket határozzon meg, ez tetszőleges típusú lehet. Amikor a függvény megváltoztatja a paramétereit, vagy a környezetét a függvény mellékhatássának nevezzük. Egyes nyelvekben van eljáráshívásra szolgáló alapszó, ez a "CALL". Egy eljárás, akkor fejeződik be szabályosan, ha elérjük a végét, vagy külön utasítással befejezzük, ez bárhol kiadható("GOTO" parancsal általában). Függvényt meghívni, csak kifejezésben lehet. Az eljárásorientált programozási nyelvekben megírt programokban kötelezően lennie kell egy speciális programegységnek, ezt főprogramnak hívjuk.

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

A blokk olyan programegység, amely csak egy másik program belsejében állhat. A blokknak van kezdete, törzse, és vége. A kezdetet és a véget egy speciális karakterszó, vagy alapszó jelzi, míg a törzsben lehetnek deklarációs és végrehajtó utasítások. A blokknak nincs paramétere, bárhol elhelyezthető, egyes nyelvekben még neve is van. Aktivizálni blokkot úgy lehet, hogy szekvenciálisan rákerül a sor, vagy "GOTO" utasítással ráugrunk a kezdetére. A blokk befejeződik, ha elértük a végét, vagy "GOTO" utasítással kilépünk belőle.

VII. heti előadás (78. oldal, a "Másodlagos belépési pontok" című rész):

Bizonyos nyelvek megengedik, hogy egy alprogramot ne csak a fejen keresztül lehessen meghívni, hanem a törzsben is ki lehessen alakítani úgynevezett másodlagos belépési pontokat. Ha az alprogramba a fejen keresztül lépünk be, akkor az alprogram teljes törzse végrehajtódk, másodlagos belépési pont használatánál a törzsnek csupán egy része hajtódik végre.

VII. heti előadás (78. oldal, a "Paraméterkiértékelés" című rész):

A paraméterkiértékelés az a folyamat, amikor egy alporgram hívásánál egymáshoz rendelődnek a formális- és aktuális paraméterek, és meghatározódnak azok az információk, amelyek a paraméterátadásnál a kommunikációra szolgálnak. A paraméterkiértékelésnél mindig a formális paraméter lista az elsődleges, de aktuális paraméterlista annyi lehet, ahányszor meghívjuk az alprogramot. A paraméterkiértékelésnek három aspektusa van.

VII. heti előadás (80. oldal, a "Paraméterátadás" című rész):

A paraméterátadás az alprogramok és más programegységek közötti komunnikáció egy formája. A paraméterátadásnál mindig van egy hívó, ez tetszőleges programegység, és egy hívott, amelyik mindig az alprogram. Vannak különböző paraméterátadási módok(érték, cím, eredmény, érték-eredmény, név és szöveg szerinti). A paraméterátadás módját több dolog is befolyásolja: A nyelv csak egyetlen paraméterátadási módot ismer(pl.: C). A formális paraméter listában explicit módon meg kell adni a paraméterátadási módot(pl.: Ada). Az aktuális és formális paraméter típusa együttesen dönti el(pl.: PL/I). A formális paraméter típusa dönti el (pl.: FORTRAN). Az alprogramok formális paramétereit három csoportra oszthatjuk: Input paraméterek, ezek segítségével az alprogram kap infirmációkat a hívótól. Output paraméterek, a hívott alprogram ad át információt a hívónak. Input-output paraméterek, az információ mindkét irányba mozog.

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

A VI. heti előadáson már volt róla szó. Ott található A blokk című rész lényege.

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

A hatáskör szinonímája a láthatóság. Egy név hatásköre a program szövegének azon részét jelenti, ahol az adott név ugyanazt a programozási eszközt hivatkozza. A név hatásköre a programegység. A programegységben a deklarált nevet, a programegység lokális nevénk nevezzük. Azt a nevet, amelyre csak a programegységben hivatkozunk(nem deklaráljuk) szabad névnek hívjuk. Hatáskörkezelésnek hívjuk azt a

DRAFT

folyamatot, amikor megállapítjuk egy név hatáskörét. Két fajtája van, a hatáskörkezelésnek az egyik a statikus, a másik pedig a dinamikus. A hatáskör mindig befelé terjed, kifelé soha. Ha egy név nem lokális egy programegységben, de onnan látható, azt globális névnek hívjuk. A globális név és a lokális név relatív fogalmak. Statikus hatáskörkezelésnél a programban szereplő összes név hatásköre a forrásszöveg alapján egyértelműen megállapítható. Dinamikus hatáskörkezelésnél, viszont a hatáskör futási időben változhat és más-más futásnál más-más lehet. Az eljárásorientált nyelvek a statikus hatáskörkezelést valósítják meg.

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

Olyan adattípus, amely megvalósítja a bezárást vagy információ rejtést. tehát ennél az adattípusnál nincs reprezentácó és művelet implementáció. 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, amit majd a fordító kezel. 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 év szerinti kötés is. A paraméterátadás változónál érték, típusnévnél név szerint történik.

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

Az I/O platform-, operációs rendszer-, implemetációfüggő. Bizonyos nyelvek nem tartalmaznak eszközt, így az implementációra bízzák a megoldást. Az I/O a programnyelvekben egy eszközrendszer, amely a perifériák kommunikációjáért felel. Az I/O közzéppontjában az állomány áll. Logikai állomány egy olyan programozási eszköz, amelynek neve van, és amelynél az absztrakt állományjellemzők attribútumként jelennek meg. A fizikai állomány operációs rendszer szintű, konkrét, a perifériákon megjelenő, az adatokat tartalmazó állomány. Egy állomány funkció szerint lehet: Input állomány, Output állomány, Input-Output állomány. Az I/O során adarok mozognak a tár és a periféria között. Kérdés, hogy az adatmozgatás közben történik-e konverzió. Ennek megfelelően kétféle adatátviteli mód létezik: folyamatos(van konverzió) vagy a bináris másnéven rekord módú(nincs konverzió). A folyamatos módú átvitelnél a tárban és a periférián eltér a reprezentáció. A nyelvekben három alapvető eszközrendszer alakult ki: formátumos módú adatátvitel, szerkesztett módú adatátvitel, listázott módú adatátvitel. A bináris adatátvitelnél az adatok a tárban és a periférián ugyanúgy jelennek meg, ez csak a háttértáraknál való kommunikációnál jöhet szóba. Az átvitel alapja itt a rekord. Ha állományokkal akarunk dolgozni, akkor a következőket kell végrehajtanunk: Deklaráció: A logikai állományt mindig deklarálni kell, el kell látni a megfelelő névvel és attribútumokkal. Összerendelés: A logikai állománynak megfeleltetünk egy fizikai állományt. Állomány megnyitása: Egy álománnyal csak akkor tudunk dolgozni, ha megnyitottuk. Feldolgozás: Ha az állományt megnyitottuk, akkor abba írhatunk, vagy olvashatunk belőle. Lezárás: A lezárás operációs rendszer rutinokat aktivizál, meszünteti a kapcsolatot a logikai és a fizikai állomány között. Az implicit input állomány a szabvány rendszerbemeneti periféria, az implicit output állomány a szabvány rendszerkimeneti periféria. C-ben az I/O eszközrendszer nem része a nyelvnek, és standar könyvtári függvények állnak rendelkezésre.

XI. heti előadás (112. oldal, a "Kivételkezelés" című rész):

A kivételkezelés lehetővé teszi, hogy az operációs rendszertől átvegyük a megszakítások kezelését. A kivételek olyan események, amelyek megszakítást okoznak. A kivételkezelés az a tevékenység, amit a program

DRAFT

végez, ha egy kivétel következik be. A kivételkezelő egy olyan programrész, amely egy adott kivétel bekövetkezése után lép működésbe. A kivételkezeléssel az eseményvezérlést teszi lehetővé a programozásban. Lehetőségünk van, akár nyelvi szinten is maszkolni a megszakításokat. Egyes kivételek figyelése letiltható vagy engedélyezhető. A kivételeknek általában van neve, és kódja is.

Milyen beépített kivételek vannak a nyelvben? Például a memóriaelérés vagy a nullával való osztás stb.

Definiálhat-e a programozó saját kivételt? Igen.

Milyenek a kivételkezelő hatáskör szabályai? Van, de ez akár lehet az egész program is.

Hogyan folytatódik a program a kivételkezelés után? Futhat tovább, de hibától függ, hogy le kell állítai(kernel megállítja) vagy le áll magáól.

Van-e a nyelvben beépített kivételkezelő? Igen van.

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

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

Alapvető eszköze a Java-nak. Használatakor létrejön egy kivétel objektum: vannak kivétel-osztályok és annak kivétel-példányai. A "JVM" feladata, hogy megkeressen egy adott objektumnak megfelelő típusú, az adott pontban látható kivételkezelőt, amely kivételkezelő az adott kivételt elkapja. Egy kivételkezelő megfelelő típusú, ha a kivételkezelő típusa megegyezik a kivétel típusával, és ha a kivételkezelő típusa őse a kivétel típusának. A láthatóságot, maga a kivételkezelő definiálja, ami egy blokk. A kivételkezelő a tetszőleges kódrészlethez köthető a JAVA-ban, és egymásba ágyazhatóak. JAVA-ban két fajta kivételt különböztetünk meg: az ellenőrzött(egy metódus láthatósági körében léphet fel), és a nem ellenőrzött(ellenőrzése vagy nagyon kényelmetlen vagy lehetetlen) kivvételt. Egy módszer fejében tehát meg kell adni, azokat az ellenőrzött kivételeket, melyeket a módszer nem kezel, de futás közben bekövetkezhetnek, ez a "THROW kivételnév_lista" utsaítás segítségével történik. A kivételek kezeléséhez a "java.lang" csomagban definiált ősosztálya "Throwable"(ezzel dobunk el) objektumai dobhatók el. Két standard alosztály van: az Error(rendszerhibák, ezek nem ellenörzöttek) és a Exception(ellenőrzött kivételek osztálya, innen származtathatunk saját ellnőrzött kivételeket). A "catch" ág teljesen hiányozhat. A típusegyeztetés miatt a felírás sorrendje nagyon lényeges, ugyanis a "catch" ág az ellenőrzött kivételt kapja el.

10.3. A C programozási nyelv by Brian W. Kernighan – Dennis M. Ritchie (K and R könyv)

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

DRAFT

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. Kifejezésutasítások, az utasítások(kifejezésutasítás, értékadás, függvényhívás) többsége ilyen. Összetett utasítás, több utasítást egyetlen utasításként kezeli, ez a fordításhoz szükséges, mivel sok fordítóprogram csak egyetlen utasítást fogad el. Kiválasztott utasítások, minden esetben a 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).

10.4. Szoftverfejlesztés C++ nyelven by Benedek Zoltán - Levendovszky Tihamér (BME C++ könyv)

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.-59.):

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 azt jelenti, 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.

DRAFT

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 struktúrák vagy akár osztályok is.

VII.heti előadás (93.-96.):

A C++ egy típustámogatott nyelv, a beépített típusokra számos művelet értelmezhető. A C nyelvben az operátorok az argumentumaikon végeznek műveletet, ennek az eredményét a visszatérítési értékük feldolgozásával használhatjuk. Egyes operátorok az argumentumaik értékét is megváltoztatják, ezt az operátor mellékhatásának nevezzük. Az operátok kiértékelésének sorrendjét zárójelekkel befolyásolhatjuk. A függvényneveket és az operátor neveket is túlterhelhetjük. A C nyelvben a függvény nem képes mellékhatásra, ez csak C++-ban lehetséges. A függvények és az operátorok különbsége a kiértékelés szabályrendszerében jelentkezik. Az operátorok speciális nevű függvények. A nem statikus tagfüggvényeknek van egy rejtett paramétere, amely megegyezik az osztály típusával.

IX. heti előadás (73.-90.):

A C nyelvben három állomány leíró áll a rendelkezésünkre: "stdin" a bemenet, "stdout" a kimenet, és az "stderr" pedig a hibakimenet. Ezek mindegyike "File" típusú. Az I/O használatához be kell építenünk az "iostream" állományt. A programban a beolvasást a "cin" -el végezzük, a kiírtaást pedig a "cout"-al. Érdemes figyelni a "nyilakra", mert az irányükk függ attól, hogy éppen be olvasunk, vagy kiíratunk. A cin állpotát beolvasás után mindig ellenőriznünk kell. Az "ignore" függvény segítségével megadhatjuk, hogy a beolvasás a sor végéig történjen, mert alapból egy írásjel(, . : stb.) megtöri a beolvasást. Megadhatjuk a maximális adatfolyam-méretét a "limits" segítségével. Így megelőzhetjük a beolvasások egymásra futását. A rendszerhívások költsége igen nagy, ezért az adatfolyamokat egy bifferrel látják el. Ezek a bufferek összegyűjtik a karaktersorozatokat, és több "cout" kiírást egy rendszerhívással írnak ki a képernyőre. A buffereket a "flush" segítségével lehet üríteni. Az adatfolyam állapotát az "iostat" típusú tagváltozó jelzi, ennek állapotát az alábbi konstansokkal lehget beállítani: eofbit(adatfolyam elérte az állomány végét), failbit(formátum hibát jelez), badbit(fatális hibát jelez), goodbit(jelzi, hogy minden rendben van). Ajelzők beállítását a "clear" tagfüggvény végzi. Nagyon fontos, hogy ha egy adatfolyam bármelyik hibabitje beállítódik, akkor az összes utána következő írási, és olvasási művelet, azon az adatfolyamon hatástalan marad(lefut, de nem tesz semmit). A C++ tartalmaz egy "string" osztályt, amely szükség szerint változtatja a méretét. Ha egy ilyen "stringet" szeretnénk beolvasni, akkor azt a "std::getline" fügvénnyel tehetjük meg, mert a szólözöknél megakadna a beolvasás. Az adatfolyam-objektumoknak vannak tagfügvényeik, amelyekkel beéllíthatjuk az állapotát(olvashatunk, írhatunk, műveleteket végezhetünk). Hasznáálhatunk még manipulátorokat is. Az I/O manipulátor egy adatfolyam-módosító speciális objektum. Vannak jelzőbitek is, olyan bitek, amelyeket bállíthatunk, vagy törölhetünk. Minden bithez tartozik egy bináris szám: ebben a bináris számban csak az a bit egyes, ahányadik biten az adott tulajdonságot beállítjuk. A C++ I/O-hoz kapcsolódó jelzőbitek az "ios" nevű osztályban vannak definiálva. Az alábbi dolgokat formázhatjuk: mezőszélesség, a kitöltő karakter, igazítás, az egész számok számrendszere, a lebegőpontos számok formátuma, mutassa-e a "+" jelet, a helykitöltő nullákat, ill. az egész számok számrendszerének alapját, kis- vagy nagybetűk. ha egyik jelzőbit sincs beállítva, akkor az adatfolyam a legjobb formázást próbálja kiválasztani egy adott számra, annak nagyságától függően. A C állománykezelése egy "FILE" típusú leíró köré csoportosul, amelyet az "fopen" függvény ad vissza. A C++ az állománykezeléshez is adatfolyamatokat használ, amelyeket ezúttal az "ifstream" (input file stream), illetve az "ofstream" (output file stream) osztályok reprezentálnak. A kétirányú adatfolyamot az "fstream" osztály valósítja meg. Az állományok megnyitását a

DRAFT

konstruktorok végzik, a lezárását pedig a destruktorok. Ha a konstruktor, vagy a destruktor nem felel meg nekünk, akkor létezik "open", illetve "close" függvény is erre a célra. Az állomány-adatfolyamosztályok ddefiníciói az "fstream" fejlécben találhatók az "std" névtérben. A jelzőbiteket az "ios::" előtaggal kell ellátni. A bemeneti adatfolyamok esetén az olvasási pozícionálást a "get", míg a kimneneti adatfolyamatok esetén az írási pozícionálást a "put" végzi. A "tell" függvények visszatérési típusa a "pos_type", amely nem egész jellegű. A pozíciókat el is tárolhatjuk, ezzeket a "seek" függvényeknek adható. A "cin", "cout", "cerr" és "clog" esetén nem használhatunk pozícionáló függvényeket. Fájlrendszerbeli állomány esetén nem válthatunk akárhogyan az olvasás és az írás között. Általában egy pozícionáló műveletet kell végeznünk.

IX. heti előadás (165.-178.):

C nyelvben az "enum" és az "int" típus között oda-vissza létezik implicit konverzió. A C++-ban viszont, ha "enum" típusra konvertálunk, akkor ki kell írnunk a típuskonverziót. A C automatikus konverziót biztosít a "void*" típusú pointer és tetszőleges típusú pointer között oda-vissza, a C++-ban ezt a konverziót is ki kell írnunk. Nem konstans referenciára nincs automatikus konverzió inkompatibilis típusok referenciáiról. Referenciát akkor is használunk, ha meg akarjuk takaírtani függvényhívásból eredő másolást. A konstans referencia alkalmazásával nagyobb objektumok átadása esetén jelentős másolási költséget takaríthaunk meg. A típuskonverziós lehetőségek két fontos esetét különböztetjük meg: az öröklés szempontjábl két független típus közti típuskonverzió, és az öröklés hierarchia mentén típuskonverzió. C++-ban ha egy másik típusról szeretnénk konvertálni a mi osztályunk típusára, akkor a konverziós konstruktor jelent megoldást. Ha az osztályunkról szertnénk egy másik típusra konvertálni, akkor a konverziós operátort érdemes használnunk. A konstruktor képes konverziót végrehajtani. A konverziók leggyakoribb hibája, hogy bizonyos kifejezések esetén több megoldás is létezik, és a fordító nem tud választani. Ilyenkor a C++ hibaüzeneteket ad ilyen esetekben. ha a típuskonverziós útvonal nem egyértelmű, akkor fordítási hibát jelent. Ha az adottt pointer, illetve referencia mögött nem olyan leszármazott van, amire konvertálunk, a viselkedés durva futási idejű hiba lesz. Az explicit típuskonverziót C nyelven a kifejezés elé () zárjólelek közé írt új típus megadásával definiálhatjuk. A C++ saját konverziós operátrokat definiál, amelyek jobban kifejezik a típuskonverzió jelentését. Az alábbi operátorok segítenek, hogy pontosabban meg tudjuk adni a konverzió célját: static_cast(statikus típuskonverzió), const_cast(konstans típuskonverzió), dynamic_cast(dinamikus típuskonverzió) és a reinterprent_cast(újraértelmező típuskonverzió). A C stílusú típuskonverzió helyett leggyakrabban a statikus típuskonverziót használjuk. A statikus típuskonverziónak megmaradtak azok a megkötései, amely a C stílusú elődjének pl.:nem konvertálhat struktúrát egész típussá vagy konstans típust nem konstans típussá. Erre van külön típuskonverzió operátor: a konstans típuskonverzió. A konstans típuskonverzió képes egyedül konstans típust nem konstanssá tenni, illetve "volatile" típust nem azzá tenni. A dinamikus típuskonverzió szintén speciális típuskonverziót valósít meg: az öröklési hierarchián lefelé történő konverziókhoz szükséges. Az újraértelmezhető típuskonverzió az implementációfüggő konverziók esetén használható. Általáb pointerre alkalmazzuk.

XI. heti előadás (187.-197.):

A kivételkezelés olyan mechanizmus, amely biztosítja, hogy ha hibát detektálunk valahol, akkor a futás a hibakezelő ágon folytatódjon. A megoldás nem csak hiba, hanem bármilyen "kivételes" helyzet esetén használható, ezért hívják kivételkezelésnek.

```
Egy példa a kivételkezelésre:

    #include <iostream>
    usning namespace std;

    int main()
    {
```

DRAFT

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

Bekérünk a felhasználótól egy nem nulla számot, majd ezt eltároljuk a "d" nevű változóban. Utána a "d" változóban eltárolt számot ellenőrizzük, hogy nem nulla e. Erre kell az "if", ha ez nulla akkor a "throw" segítségével kidobjuk, mint lehetséges hibát. Ha nem nulla, akkor meghatározzuk a reciprokát, és kiíratjuk. A "catch" részben elkapjuk a "throw" által eldobott hibát, és kiírjuk, az "if"-ben megadott mondatot. Mindkét megoldás végén a program kiírja, hogy "Done.".

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

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

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

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

A "try-catch" blokkok egymásba is ágyazhatóak. Így lehetőségünk van arra, hogy bizonyos kivételeket a dobott kivételhez közel, alacsonyabb szinten kezeljünk. Az elkapott kivételet a "throw" kulcsszó paraméter nélküli alkalmazásával újradobható. Egy kivétel dobásakkor annak elkapásáig a függvények hívási láncában felfelé haladva az egyes függvények lokális változói felszabadulnak. Ezt a folyamatot a hívási verem visszacsévélésnek nevezzük.

```
A verem visszacsévélése:

    int main()
    {
     try
     {
       f1();
     }
```

```
     catch(const char* errorText)
     {
       cerr << errotext << endl;
     }
    }

    void f1()
    {
      Fifo fifo; //a fifo egy általunk megírt osztály
      f2();
      ...
    }

    void f2()
    {
      int i = 1;
      throw "error1";
    }
```

A lépések a példában:

Először az "f2" kivételt dob, ezután az "f2"-ben definiált "i" lokális változó felszabadul. Majd az "f1"-ben lefoglalt "Fifo fifo" objektum felszabadul, meghívódik a destruktorra. Végül pedig lefut a "main" függvényben lévő "catch" blokk.

A kivétel elkapása, és dobása között futhat le kód, mivel meghívódnak a verem visszacsévélése során felszabadított objektumok destruktorai. Fontos, hogy a kivétel dobása, és elkapása között ne dobjunk újabb kivételt, mert azt már nem lehet kezelni.

XI. heti előadás (211.):

```
Erőforrás kezelés:

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

```
            delete pMessage;
            throw;
          }
        }
      }
    private:
      Message* readNextMessage(istream& is)
      {   ... }
    };
```

Példa magyarázata:

A "MessageHandler" egy bemeneti folyamból üzeneteket kiolvasó, és feldolgozó osztály. A "ProcessMessage" tagfüggvénye mindaddig "Message" objektumokat olvas a bemeneti folyamból a "readNextMessage" függvény meghívásával, amíg az "NULL"-al nem tér vissza. "readNextMessage" működése: a new operátorral létrehoz egy "Message"-t, a hozzá tartozó adatokat kiolvassa a folyamból, és visszatér a "Message" objektumra mutató pointerrel. A "ProcessMessage" a "readNextMessage" hívását követően olyan függvényeket hív, melyek feldolgozzák a beolvasott üzenetet, ezt követően pedig a "delete" operátorral felszabadítjuk a "Message" objektumot. Probléma akkor van, ha az üzenet a feldolgozást végző fügvények kivételt dobnak, mert így akkor nem tudjuk felszabadítani az utoljára beolvasott "Message" objektumot. Erre a problémára a megoldás a "try-catch", mivel az üzenet feldolgozása során bármilyen kivétel keletkezik, a "catch"-el elkapjuk, felszabadítjuk a helyileg lefoglalt memóriát, majd újradobjuk a kivételt. A kivétel újradobása nagyon fonos, hiszen ha ezt kihagyjuk, akkor a hiba rejtve marad.

DRAFT

III. rész

Második felvonás

DRAFT

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

DRAFT

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

IV. rész

Irodalomjegyzék

DRAFT

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.

DRAFT