Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert	2019. március 21.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés 1

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

II. Tematikus feladatok 3

2. Helló, Turing! 5
2.1. Végtelen ciklus . . . 5
2.2. Lefagyott, nem fagyott, akkor most mi van? . . . 6
2.3. Változók értékének felcserélése . . . 8
2.4. Labdapattogás . . . 9
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . 10
2.6. Helló, Google! . . . 12
2.7. 100 éves a Brun tétel . . . 13
2.8. A Monty Hall probléma . . . 14

3. Helló, Chomsky! 17
3.1. Decimálisból unárisba átváltó Turing gép . . . 17
3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen . . . 17
3.3. Hivatkozási nyelv . . . 17
3.4. Saját lexikális elemző . . . 18
3.5. l33t.l . . . 18
3.6. A források olvasása . . . 18
3.7. Logikus . . . 19
3.8. Deklaráció . . . 19

4. Helló, Caesar! 21
4.1. int ** háromszögmátrix . . . 21
4.2. C EXOR titkosító . . . 22
4.3. Java EXOR titkosító . . . 23
4.4. C EXOR törő . . . 24
4.5. Neurális OR, AND és EXOR kapu . . . 27
4.6. Hiba-visszaterjesztéses perceptron . . . 27

5. Helló, Mandelbrot! 34
5.1. A Mandelbrot halmaz . . . 34
5.2. A Mandelbrot halmaz a `std::complex` osztállyal . . . 34
5.3. Biomorfok . . . 34
5.4. A Mandelbrot halmaz CUDA megvalósítása . . . 34
5.5. Mandelbrot nagyító és utazó C++ nyelven . . . 34
5.6. Mandelbrot nagyító és utazó Java nyelven . . . 35

6. Helló, Welch! 36
6.1. Első osztályom . . . 36
6.2. LZW . . . 36
6.3. Fabejárás . . . 36
6.4. Tag a gyökér . . . 36
6.5. Mutató a gyökér . . . 37
6.6. Mozgató szemantika . . . 37

7. Helló, Conway! 38
7.1. Hangyaszimulációk . . . 38
7.2. Java életjáték . . . 38
7.3. Qt C++ életjáték . . . 38
7.4. BrainB Benchmark . . . 39

8. Helló, Schwarzenegger! 40
8.1. Szoftmax Py MNIST . . . 40
8.2. Szoftmax R MNIST . . . 40
8.3. Mély MNIST . . . 40
8.4. Deep dream . . . 40
8.5. Robotpszichológia . . . 41

9. Helló, Chaitin! 42
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . 42
9.2. Weizenbaum Eliza programja . . . 42
9.3. Gimp Scheme Script-fu: króm effekt . . . 42
9.4. Gimp Scheme Script-fu: név mandala . . . 42
9.5. Lambda . . . 43
9.6. Omega . . . 43

III. Második felvonás 44

10. Helló, Arroway! 46
10.1. A BPP algoritmus Java megvalósítása . . . 46
10.2. Java osztályok a Pi-ben . . . 46

11. Helló, Gutenberg! 47
11.1. Programozási alapfogalmak . . . 47
11.2. Programozás bevezetés . . . 47
11.3. Programozás . . . 49

IV. Irodalomjegyzék 50
11.4. Általános . . . 51
11.5. C . . . 51
11.6. C++ . . . 51
11.7. Lisp . . . 51

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

A következő program egy magot 100 százalékban dolgoztat meg:

```
#include <stdio.h>

int main()
{
    while(1){};
}
```

Ez a program egy egyszerű while ciklus, aminek ha a feltétele teljesül, akkor nem csinál semmit. Ebben az esetben a feltétel 1, ami 'igaz' vagy angolul 'true' boolean típusú értékkel bír, tehát az adott ciklus egy végtelen ciklus, azaz sosem áll le, kivéve ha arra kényszerítjük a CTRL+C betűkombinációval, hiszen az 'igaz' értéke mindig igaz.

A következő program minden magot 100 százalékban dolgoztat meg:

```
        ]
// compile: gcc name.c -fopenmp

#include <stdio.h>

int main()
{
    #pragma omp parallel
    while(1){};
}
```

Ez az előző program módosítása. Ugyanabból a while ciklusból áll némi változtatással. A #pragma feladata, hogy a gép vagy az operációs rendszerre vonatkozó specifikus utasítást adjon a compilernek, azaz megmondja hogy a compiler tegyen valamit vagy felülírjon valamilyen általánosnak vett utasítást. Itt a #pragma szerepe a felülírás. Normális esetben, mint azt felül is láthattuk, ez a program #pragma nélkül csak 1 processzort dolgoztat meg 100 százalékon, de az omp parallel utsítás azt mondja a compilernek, hogy egyszerre több szálon fusson a program, azaz több processzor dolgozzon rajta egy időben, aminek az eredménye, hogy ez a program az összes processzort 100 százalékosan megdolgoztatja.

A következő program 0 százalékon dolgoztat meg egy magot:

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

        int main()
        {
            while(1)
            {
                usleep(1000);
            }
        }
```

Ez a program a while ciklusban a usleep(1000) utasítást végzi el. A usleep(seconds_t usec) az unistd.h fájlban definiált függvény. Adott usec mennyiségű mikroszekundumig megszakítja a program működését.

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!

Megoldáás videó:

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

```
Program T100
{

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

  main(Input Q)
```

```
  {
    Lefagy(Q)
  }
}
```

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

- Ha T1000 lefagyó, akkor nem fog lefagyni, kiírja, hogy true

- Ha T1000 nem fagyó, akkor pedig le fog fagyni...

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

A megoldási probléma lényege hogy nem tudunk olyan programot írni, amely eldönti egy másik programról, hogy az lefagy-e vagy sem. Ahhoz hogy a T100 eldöntse, hogy egy program lefagy-e vagy sem, elősször le kell futtatnia azt és megvizsgálni a következményeket. Ha a prgoram lefutott, a T100 kiírja hogy a programunk lefutott, de ha a program lefagy, akkor a T100 is le fog fagyni, mert a döntés előtt le kell futtatnia azt. Ha a fenti T1000 programot önmagára futtatjuk, akkor két eshetőség adódik: Ha a programunk nem fagy le, akkor bekövetkezik a végtelen ciklus a T1000-ben és így lefagy. Ha pedig lefagy a programunk, akkor mivel a program lefagy ezért a T1000 is lefagy.

2.3. Változók értékének felcserélése

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

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

```
#include <stdio.h>

int main()
{
    int a = 10;
    int b = 4;
    printf("a = %d\n", a);
    printf("b = %d\n", b);

    a = a - b;  a = 10 - 4 / 6
    b = a + b;  b = 6 + 4 / 10
    a = b - a;  a = 10 - 6 / 4

    printf("a = %d\n", a);
    printf("b = %d\n", b);
}
```

A fenti C program kiírja két változó értékét, felcseréli őket, majd kiírja a változók új értékeit. Az int a = 10 és int b = 4 megadják a és b értékeit. Először a-t egyenlővé tesszük a - b -vel (itt így a = 10 - 4 = 6). Majd b-t egyenlővé tesszük a + b -vel, de mivel a = a - b, ezért b = a + b = a - b + b = a (b = 10 - 4 + 4 = 10). Ekkor a-t egyenlővé tesszük b - a -val. Itt b = a és a = a - b -ként van definiálva, tehát a = b - a = a - (a - b) = a - a + b = b (a = 6 + 4 - (10 - 4) = 4) A printf() függvény kiírja a paraméteréül kapott stringet. Például a fent látható printf("a = %d\n", a) azt fogja kiírni hogy a = 10, majd egy új sort kezd. Ebből "a = %d\n" a string rész, amiben a %d egy format specifier. Feladata, hogy megmondja, a %d helyére egy decimális egész szám (itt: a) fog kerülni, amit a string rész utáni vesszővel elválasztott helyre kell írni. Ugyanitt a \n egy escape sequence, ami azt mondja a printf függvénynek, hogy a \n helyére egy új sort írjon.

Ezek után megkezdődik a változók felcserélése a fenti műveletek segítségével. Miután az kész, a program ismét kiírja a változók immár új felcserélt értékeit.

2.4. Labdapattogás

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

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

Labdapattogtatás if nélkül:

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

int write_x(x,y)
{
    int xi,yi;

    for(xi=0;xi<x;xi++)
    {
        printf("\n");
    }

    for(yi=0;yi<y;yi++)
    {
        printf(" ");
    }
    printf("X\n");
    return 0;
}

int main()
{

    int width = 90;
    int height = 25;
    long int x=0,y=0;

    while(1)
    {
        system("cls");
        write_x(abs(height - (x++ % (height * 2 ))), abs(width - (y ↩
            ++ % (width*2))));
        usleep(10);
    }

    return 0;
```

```
        }
```

A program Windows alatt egy adott 'pályán' pattogtat egy x-el jelölt 'labdát' a konzolon. Két függvényből áll, a már megszokott int main() és egy int write_x(x,y) függvényből. A main függvényben meg van adva a pálya hossza, magassága és a labda koordinátái, amelyek alapesetben nullával egyenlőek. Ezek után a program egy végtelen ciklusban először törli a képernyőt a system("cls") függvénnyel, majd meghívja a write_x() függvényt két abszolút értékre. Egy szám abszolút értékét az abs() függvénnyel számoljuk ki, ami a math.h fájlban van meghatározva. Ha a write_x() lefutott, akkor egy ideig megszűnik a munkavégzés és előről kezdődnek a ciklusban definiált lépések.

Az int write_x(x,y) itt egy a felhasználó által megadott függvény. Az előtte lévő int szó azt jelenti, hogy a függvény egy integer, azaz egész szám típusú értéket fog visszaadni, amit a return 0 paranccsal teszünk meg (return 0 azt jelenti, hogy a program probléma nélkül lefutott. Ha nem nullát ad vissza, akkor futás közben hiba történt.). A zárójelek közötti x és y számok a függvény paraméterei, tehát ezekkel fog dolgozni. A számokat függvényhívásnál kell megadni. A függvényen belül két for ciklus található. Az első feladata, hogy a labda függőleges helyzetének megfelelő új sort írjon ki, a másodiké pedig, hogy úgyanúgy a labda vízszintes pozíciójának megfelelő szóközt írasson ki. Ha ez megtörtént, akkor a labda jelenlegi koordinájaihoz érkezett a kurzor. A for ciklusok befejeződnek és printf()-el kiíratunk egy X-et.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása:

```
            // BHAX BogoMIPS
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
```

```
// This program is based on
//
// - Linus Torvalds's original code (https://mirrors.edge.kernel.org/pub/ ↩
   linux/kernel/v1.0/linux-1.0.tar.gz init/main.c)
// - and Jeff Tranter's standalone version (archive.debian.org/debian/pool/ ↩
   main/s/sysutils/sysutils_1.3.8.5.1.tar.gz).
//
// See also UDPROG
//

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

void
delay (unsigned long long loops)
{
  for (unsigned long long i = 0; i < loops; i++);
}

int
main (void)
{
  unsigned long long loops_per_sec = 1;
  unsigned long long ticks;

  printf ("Calibrating delay loop..");
  fflush (stdout);

  while ((loops_per_sec <<= 1))
    {
      ticks = clock ();
      delay (loops_per_sec);
      ticks = clock () - ticks;

      if (ticks >= CLOCKS_PER_SEC)
  {
    loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;

    printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / 500000,
      (loops_per_sec / 5000) % 100);

    return 0;
  }
    }

  printf ("failed\n");
  return -1;
}
```

2.6. Helló, Google!

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

Megoldás videó:

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

void kiir (double tomb[], int db);
double tavolsag(double pagerank[], double pagerank_temp[], int db);

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 PR[4] = {0.0, 0.0, 0.0, 0.0};
  double PRv[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};

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

    for (int i=0; i<4; i++)
        {
      double temp = 0.0;

      for (int j=0; j<4; j++)
            {
        temp += L[i][j] * PR[j];
        PRv[i] = temp;
      }
    }

    if(tavolsag(PR, PRv, 4) < 0.000001)
        {
      break;
    }
```

```
  kiir(PR, 4);

  return 0;
}

void kiir (double tomb[], int db)
{
  for (int i=0; i<db; i++)
  {
    printf("PageRank [%d]: %lf\n", i, tomb[i]);
  }
}

double tavolsag(double pagerank[], double pagerank_temp[], int db)
{
  double dist = 0.0;

  for(int i=0; i<db; i++)
  {
    dist += (pagerank[i] - pagerank_temp[i]) * (pagerank[i] - pagerank_temp ↩
        [i]);
  }

  return sqrt(dist);
}
```

Ez a program egy 4 honlapból álló hálózatra kiszámolja a négy lap Page-Rank értékét.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

```
#   Copyright (C) 2019  Dr. Norbert Bátfai, nbatfai@gmail.com
#
#   This program is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
```

```
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>

library(matlab)

stp <- function(x){

    primes = primes(x)
    diff = primes[2:length(primes)]-primes[1:length(primes)-1]
    idx = which(diff==2)
    t1primes = primes[idx]
    t2primes = primes[idx]+2
    rt1plust2 = 1/t1primes+1/t2primes
    return(sum(rt1plust2))
}

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

A Brun tétel szerint az ikerprímek reciprokösszege egy Brun-konstans nevű értékhez konvergál. Az ikerprímek olyan prímpárok, melyek különbsége 2. Egy prímpár pedig egy olyan prím, ami egy másik prímtől 2-nél nagyobb, vagy kisebb. Példaképp az 1 és a 3 számok. Ekkor 1/1 + 1/3 = 1.333 . A Brun-konstans megközelítő értéke 1,90216 . Itt a program első felében a primes(x) függvényt definiáljuk, ami egyrészt kiírja x-ig a prímszámokat, másrészt kiszámolja az ikerprímek reciprokösszegeit. Azután a második részben kiplottolhatjuk a függvény által kiszámolt eredményeket. Viggo Brun, norvég matematikus bizonyította be a tételt 1919-ben.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

```
#    An illustration written in R for the Monty Hall Problem
#    Copyright (C) 2019  Dr. Norbert Bátfai, nbatfai@gmail.com
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
```

```
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>
#
#    https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben ↩
    _a_monty_hall-paradoxon_kapcsan
#

kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
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)
```

A Monty Hall probléma keretében arról van szó, hogy adott három lehetőség (doboz, ajtó stb.), ebből egy nyerő és kettő nem. Nekünk a feladatunk hogy ezekből válasszunk egyet. Ilyenkor az esély hogy a nyerő dobozt választjuk egy a háromból. Választásunk után egy második fél kinyit a maradék két dobozból egy olyat, ami biztosan nem nyerő, majd felteszi nekünk a kérdést, hogy maradunk-e az előző döntésünknél, vagy változtatunk rajta. Ekkor a helyes válasz az, hogy változtassunk, ugyanis így 2/3 esélyünk van ara, hogy jó dobozt választunk szemben az előző 1/3-al.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

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...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciája

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

Mit vezetnek be a programba a következő nevek?

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. int ** háromszögmátrix

Megoldás videó:

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

int
main ()
{
    int nr = 5;
    double **tm;

    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
    {
        return -1;
    }

    for (int i = 0; i < nr; ++i)
    {
        if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL ↩
           )
        {
            return -1;
        }

    }

    for (int i = 0; i < nr; ++i)
        for (int j = 0; j < i + 1; ++j)
            tm[i][j] = i * (i + 1) / 2 + j;

    for (int i = 0; i < nr; ++i)
    {
```

```
        for (int j = 0; j < i + 1; ++j)
            printf ("%f, ", tm[i][j]);
        printf ("\n");
    }

    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0;  // mi van, ha itt hiányzik a külső ()
    *(tm[3] + 2) = 44.0;
    *(*(tm + 3) + 3) = 45.0;

    for (int i = 0; i < nr; ++i)
    {
        for (int j = 0; j < i + 1; ++j)
            printf ("%f, ", tm[i][j]);
        printf ("\n");
    }

    for (int i = 0; i < nr; ++i)
        free (tm[i]);

    free (tm);

    return 0;
}
```

Ez a program a malloc() és a free() függvényeket felhasználva helyet foglal egy alsó háromszög mátrixnak a szabad tárban, majd szabaddá teszi azt a helyet. A malloc() függvény feladata a megadott számú bájtoknak való memóriablokk lefoglalása. Visszatérítési értéke egy void típusú pointer. Ebben a programban két malloc() függvény van, mindkettő egy egy if függvényben. Ez azért van, hogyha az allokáció sikertelen, tehát nullpointer a visszatérési értéke, akkor a program befejeződjön. A malloc() által foglalt memóriaterület magától nem fog felszabadulni, ezért ezt a program végén nekünk kell megtenni a free() függvénnyel.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

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

#define MAX_KULCS 100
#define BUFFER_MERET 256
```

```
        int
        main (int argc, char **argv)
        {

        char kulcs[MAX_KULCS];
        char buffer[BUFFER_MERET];

        int kulcs_index = 0;
        int olvasott_bajtok = 0;

        int kulcs_meret = strlen (argv[1]);
        strncpy (kulcs, argv[1], MAX_KULCS);

        while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
            {
            for (int i = 0; i < olvasott_bajtok; ++i)
          {

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

          }

            write (1, buffer, olvasott_bajtok);

            }
        }
```

A XOR kódoló mögötti alapötlet, hogy a fájl bájtjait össze XOR-ozzuk a kulcs bájtjaival. Ekkor az eredmény egy titkosított fájl, aminek az eredetijét visszakaphatjuk, ha az eredményt ismét össze XOR-ozzuk a kulccsal. A XOR jelentése kizáró vagy, elvégezni az ^ operátorral lehet. A XOR B eredménye csak akkor igaz, ha vagy csak A, vagy csak B igaz. Minden más esetben hamis. Tehát: 1 ^ 1 = 0 0 ^ 1 = 1 1 ^ 0 = 1 0 ^ 0 = 0

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

```
public class ExorTitkosító {

    public ExorTitkosító(String kulcsSzöveg,
            java.io.InputStream bejövőCsatorna,
            java.io.OutputStream kimenőCsatorna)
            throws java.io.IOException {
```

```
        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBájtok = 0;

        while((olvasottBájtok =
                bejövőCsatorna.read(buffer)) != -1) {

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

                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
                kulcsIndex = (kulcsIndex+1) % kulcs.length;

            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);

        }

    }

    public static void main(String[] args) {

        try {

            new ExorTitkosító(args[0], System.in, System.out);

        } catch(java.io.IOException e) {

            e.printStackTrace();

        }

    }

}
```

Ez az előző feladatnak a Java nyelven megvalósított változata.

4.4. C EXOR törő

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

A következő program megtöri a titkosított fájlunkat.

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
```

```c
#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)
{

    int kulcs_index = 0;

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

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

    }

}

int
```

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

    exor (kulcs, kulcs_meret, titkos, titkos_meret);

    return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{

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

```
                                    if (exor_tores (kulcs, KULCS_MERET,  ↩
                                       titkos, p - titkos))
                                        printf
                                        ("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta ↩
                                            szoveg: [%s]\n",
                                         ii, ji, ki, li, mi, ni, oi, pi,  ↩
                                            titkos);

                                    // ujra EXOR-ozunk, igy nem kell egy  ↩
                                       masodik buffer
                                    exor (kulcs, KULCS_MERET, titkos, p -  ↩
                                       titkos);
                                }

    return 0;
}
```

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat...

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

```
class Perceptron
{
public:
  Perceptron ( int nof, ... )
  {
    n_layers = nof;

    units = new double*[n_layers];
    n_units = new int[n_layers];
```

```
    va_list vap;

    va_start ( vap, nof );

    for ( int i {0}; i < n_layers; ++i )
      {
        n_units[i] = va_arg ( vap, int );

        if ( i )
          units[i] = new double [n_units[i]];
      }

    va_end ( vap );

    weights = new double**[n_layers-1];

#ifndef RND_DEBUG
    std::random_device init;
    std::default_random_engine gen {init() };
#else
    std::default_random_engine gen;
#endif

    std::uniform_real_distribution<double> dist ( -1.0, 1.0 );

    for ( int i {1}; i < n_layers; ++i )
      {
        weights[i-1] = new double *[n_units[i]];

        for ( int j {0}; j < n_units[i]; ++j )
          {
            weights[i-1][j] = new double [n_units[i-1]];

            for ( int k {0}; k < n_units[i-1]; ++k )
              {
                weights[i-1][j][k] = dist ( gen );
              }
          }
      }
  }

  Perceptron ( std::fstream & file )
  {
    file >> n_layers;

    units = new double*[n_layers];
    n_units = new int[n_layers];

    for ( int i {0}; i < n_layers; ++i )
      {
```

```
        file >> n_units[i];

        if ( i )
          units[i] = new double [n_units[i]];
      }

    weights = new double**[n_layers-1];

    for ( int i {1}; i < n_layers; ++i )
      {
        weights[i-1] = new double *[n_units[i]];

        for ( int j {0}; j < n_units[i]; ++j )
          {
            weights[i-1][j] = new double [n_units[i-1]];

            for ( int k {0}; k < n_units[i-1]; ++k )
              {
                file >> weights[i-1][j][k];
              }
          }
      }
  }


  double sigmoid ( double x )
  {
    return 1.0/ ( 1.0 + exp ( -x ) );
  }


  double operator() ( double image [] )
  {

    units[0] = image;

    for ( int i {1}; i < n_layers; ++i )
      {

#ifdef CUDA_PRCPS

        cuda_layer ( i, n_units, units, weights );

#else

        #pragma omp parallel for
        for ( int j = 0; j < n_units[i]; ++j )
          {
            units[i][j] = 0.0;
```

```
            for ( int k = 0; k < n_units[i-1]; ++k )
              {
                units[i][j] += weights[i-1][j][k] * units[i-1][k];
              }

            units[i][j] = sigmoid ( units[i][j] );

          }

#endif

      }

    return sigmoid ( units[n_layers - 1][0] );

  }

  void learning ( double image [], double q, double prev_q )
  {
    double y[1] {q};

    learning ( image, y );
  }

  void learning ( double image [], double y[] )
  {
    //( *this ) ( image );

    units[0] = image;

    double ** backs = new double*[n_layers-1];

    for ( int i {0}; i < n_layers-1; ++i )
      {
        backs[i] = new double [n_units[i+1]];
      }

    int i {n_layers-1};

    for ( int j {0}; j < n_units[i]; ++j )
      {
        backs[i-1][j] = sigmoid ( units[i][j] ) * ( 1.0-sigmoid ( units[i][ ↩
            j] ) ) * ( y[j] - units[i][j] );

        for ( int k {0}; k < n_units[i-1]; ++k )
          {
            weights[i-1][j][k] += ( 0.2* backs[i-1][j] *units[i-1][k] );
          }

      }
```

```
  for ( int i {n_layers-2}; i >0 ; --i )
    {

      #pragma omp parallel for
      for ( int j =0; j < n_units[i]; ++j )
        {

          double sum = 0.0;

          for ( int l = 0; l < n_units[i+1]; ++l )
            {
              sum += 0.19*weights[i][l][j]*backs[i][l];
            }

          backs[i-1][j] = sigmoid ( units[i][j] ) * ( 1.0-sigmoid ( units ↩
             [i][j] ) ) * sum;

          for ( int k = 0; k < n_units[i-1]; ++k )
            {
              weights[i-1][j][k] += ( 0.19* backs[i-1][j] *units[i-1][k]  ↩
                 );
            }
        }
    }

  for ( int i {0}; i < n_layers-1; ++i )
    {
      delete [] backs[i];
    }

  delete [] backs;

}

~Perceptron()
{
  for ( int i {1}; i < n_layers; ++i )
    {
      for ( int j {0}; j < n_units[i]; ++j )
        {
          delete [] weights[i-1][j];
        }

      delete [] weights[i-1];
    }

  delete [] weights;

  for ( int i {0}; i < n_layers; ++i )
```

```
      {
        if ( i )
          delete [] units[i];
      }

    delete [] units;
    delete [] n_units;

  }

  void save ( std::fstream & out )
  {
    out << " "
        << n_layers;

    for ( int i {0}; i < n_layers; ++i )
      out << " " << n_units[i];

    for ( int i {1}; i < n_layers; ++i )
      {
        for ( int j {0}; j < n_units[i]; ++j )
          {
            for ( int k {0}; k < n_units[i-1]; ++k )
              {
                out << " "
                    << weights[i-1][j][k];

              }
          }
      }

  }

private:
  Perceptron ( const Perceptron & );
  Perceptron & operator= ( const Perceptron & );

  int n_layers;
  int* n_units;
  double **units;
  double ***weights;

};
```

A fenti osztály egy hiba-visszaterjesztéses perceptron definíciója. Az így definiált perceptronnak helyet foglalhatunk úgy, hogy Perceptron* p = new Perceptron(int layer, int neuron1, int neuron2, ..., int neuronN) Ahol a layer a rétegek számát jelöli és az azutáni egész számok pedig az egyes rétegekhez rendelt neuronok

számát.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

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

Megoldás videó:

Megoldás forrása:

5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

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

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

9.3. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.4. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. Omega

Megoldás videó:

Megoldás forrása:

III. rész

Második felvonás

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

10. fejezet

Helló, Arroway!

10.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.2. 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...

11. fejezet

Helló, Gutenberg!

11.1. Programozási alapfogalmak

[?]

11.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

The C Programming Language, Brian W. Kernighan, Dennis M. Ritchie

1.1 Indulás.

A könyv írói szerint minden program elsajátítása a "Helló Világ!" program megírásával történik. Ez C nyelven így néz ki:

```
include <stdio.h>

int main()
{
    printf("Hello World!"\n);
}
```

Ezt a forráskódot programnev.c néven kell elmenteni, futtatása operációs rendszertől függ. Lefordítása a "gcc programnev.c -nev" paranccsal történik, futtatása Linuxon "./nev" , Windows alatt "nev" a terminálból. A program eredménye a következőképpen fog kinézni: Hello World!

Egy C program függvényekből (function) és változókból (variable) áll. Egy függvény pedig állításokból (statement). Az állítások megmondják, hogy mit kell csinálni, a változók pedig a számításokhoz szükséges

értékeket tárolják. A munkánkat különböző könyvtárak segíthetik, amelyeket inkludálni kell. Ezt itt az első sor teszi. A legelső sor azt mondja a compilernek, hogy az stdio.h (standard input/output) fájl tartalmát másolja át a mi programfájlunkba.

Az argumentumok azok az értékek, amelyeket függvényhíváskor adunk meg. Például a "printf("Helló Világ!\n")" esetén az printf() függvény argumentuma a "Helló Világ!\n" karakterlánc. Az argumentumokat mindig egy zárójelbe tesszük.

1.2 Változók és aritmetikai kifejezések

A következő program két oszlopban kiírja nullától háromszázig huszassával haladva a hőmérsékletet fahrenheit fokban mérve, és mellé celsiusban a C = (5/9)(F-32) képletet használva, ahol a C a celsius, F a fahrenheitban mért hőmérséklet.

```
#include <stdio.h>

/* fahrenheit celsius tabla kiíratasa
fahr = 0, 20, ..., 300-ra */
main()
{
  int fahr, celsius;
  int lower, upper, step;

  lower = 0;   /* minimum homerseklet */
  upper = 300;   /* maximum homerseklet */
  step = 20;   /* lepesek nagysaga */

  /* 20-al noveljuk fahr erteket a maximum elereseig */
  fahr = lower;
  while (fahr <= upper) {
    celsius = 5 * (fahr - 32) / 9;
    printf("%d\t%d\n", fahr, celsius);
    fahr = fahr + step;
  }
}
```

A változókat, használatuk előtt, mindig deklarálni kell a "típus név" formában. Ezeket deklarációknak hívjuk. A C nyelvben több különböző változótípust ismerünk, mint az int (integer - valósszám), char (character - karakter), float (floating point - lebegőpontos szám). A változók méretét tekintve is beszélhetünk mutatókról (pointer), tömbökről (array), uniókról (union), vagy struktúrákról (structure)

A program azzal kezdődik, hogy elvégezzük a szükséges deklarációkat, illetve értékeket adunk a step, lower és upper értékeknek. Ezután következik egy while loop, miszerint amíg a fahr kisebb vagy egyenlő az upper számnál, addig számolja ki a fahrhoz tartozó celsius értéket és írassa ki. Végül növeljök a fahr értékét steppel.

A while loopon belül láthatjuk, hogy a printf() függvényt használhatjuk az output formázására is. A %d decimális egész szám helyét jelöli, melyeket a kiírandó karakterlánc után adunk meg. Azt, hogy milyen

értéket akarunk kiíratni a % jel utáni betűk és számok kombinációjával tudjuk meghatározni. Például %6d (minimum 6 karakter széles decimális integer), %6.2f (minimum 6 karakter széles, tizedes vessző után két számjeggyel rendelkező lebegőpontos szám). Ugyanitt láthatunk példát escape sequence-re is (/n és /t). Az escape sequence-ek olyan / után írt karakterek, amelyeknek speciális funkcióik vannak. Például a /n új sort ír, a /t pedig egy tabulátort.

11.3. Programozás

[BMECPP]

IV. rész

Irodalomjegyzék

11.4. Általános

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

11.5. C

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

11.6. C++

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

11.7. 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.