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 Ã©s Rácz, Ferenc	2019. március 19.	

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 . . . 7
2.4. Labdapattogás . . . 8
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . 8
2.6. Helló, Google! . . . 8
2.7. 100 éves a Brun tétel . . . 10
2.8. A Monty Hall probléma . . . 10

3. Helló, Chomsky! 13
3.1. Decimálisból unárisba átváltó Turing gép . . . 13
3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen . . . 13
3.3. Hivatkozási nyelv . . . 14
3.4. Saját lexikális elemző . . . 14
3.5. l33t.l . . . 14
3.6. A források olvasása . . . 14
3.7. Logikus . . . 16
3.8. Deklaráció . . . 16

4. Helló, Caesar! 18
4.1. int *** háromszögmátrix . . . 18
4.2. C EXOR titkosító . . . 19
4.3. Java EXOR titkosító . . . 20
4.4. C EXOR törő . . . 20
4.5. Neurális OR, AND és EXOR kapu . . . 21
4.6. Hiba-visszaterjesztéses perceptron . . . 22

5. Helló, Mandelbrot! 23
5.1. A Mandelbrot halmaz . . . 23
5.2. A Mandelbrot halmaz a `std::complex` osztállyal . . . 23
5.3. Biomorfok . . . 23
5.4. A Mandelbrot halmaz CUDA megvalósítása . . . 23
5.5. Mandelbrot nagyító és utazó C++ nyelven . . . 23
5.6. Mandelbrot nagyító és utazó Java nyelven . . . 24

6. Helló, Welch! 25
6.1. Első osztályom . . . 25
6.2. LZW . . . 25
6.3. Fabejárás . . . 25
6.4. Tag a gyökér . . . 25
6.5. Mutató a gyökér . . . 26
6.6. Mozgató szemantika . . . 26

7. Helló, Conway! 27
7.1. Hangyaszimulációk . . . 27
7.2. Java életjáték . . . 27
7.3. Qt C++ életjáték . . . 27
7.4. BrainB Benchmark . . . 28

8. Helló, Schwarzenegger! 29
8.1. Szoftmax Py MNIST . . . 29
8.2. Szoftmax R MNIST . . . 29
8.3. Mély MNIST . . . 29
8.4. Deep dream . . . 29
8.5. Robotpszichológia . . . 30

9. Helló, Chaitin! 31
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . 31
9.2. Weizenbaum Eliza programja . . . 31
9.3. Gimp Scheme Script-fu: króm effekt . . . 31
9.4. Gimp Scheme Script-fu: név mandala . . . 31
9.5. Lambda . . . 32
9.6. Omega . . . 32

III. Második felvonás 33

10. Helló, Arroway! 35
10.1. A BPP algoritmus Java megvalósítása . . . 35
10.2. Java osztályok a Pi-ben . . . 35

IV. Irodalomjegyzék 36
10.3. Általános . . . 37
10.4. C . . . 37
10.5. C++ . . . 37
10.6. Lisp . . . 37

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:Github 2.1 megoldása

Tanulságok, tapasztalatok, magyarázat: A feladat első része egész egyszerűen megoldható. Egy olyan program ami 0 százalékban használ egy magot gyakorlatilag egy olyan program ami nem csinál semmit. Mivel nem szabad hogy véget érjen, így egy végtelen cikluson belül kell megoldani a semmittevést a következő pontosan ezt is éri el:

```
Program 0
  while (a==1){

    sleep(1);};
```

100 száazélkban leterhelni egy magot a következőképp lehet. Egy végtelen számolási ciklusba kell kényszeríteni kivezető út nélkül. Például a következőképp:

```
Program 100
  while (a==1){
    b+1;
};
```

(Az optimalizálás alapjai) Az összes magot 100 százalékban leterhelni innen már nem nehéz. Ha egyre már meg van, csak annyi a teendő hogy az összes többire magra is rábízok egy végtelen számolást, akár ugyanazt.

```
Program x*100
  #pragma omp parallel
        while (a==1){
           b+1;};
```

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

Tanulságok, tapasztalatok, magyarázat: A feladat egy olyan program írása volt amely bármely más programról eldönti hogy lefagy-e vagy sem. Ez elméletben ellentmondásokhoz vezet. Ha eltávolítjuk az ellentmondásokat, akkor csak a feladatra nem adunk választ. Azaz, a program mondjuk kapja meg saját magát "bármely program címszó alatt" bementeként. Itt jön ki az ellentmondás és az el nem távolíthatósága. Ha azt mondja magáról hogy nincs magában végtelen ciklus akkor végtelen ciklusba kerül, ha pedig van benne akkor megfagy. Emiatt nem lehetséges a feladat megoldása.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: Github 2.3 megoldása

Tanulságok, tapasztalatok, magyarázat: Ehhez fontos tudni hogy a számítógép hogyan is működik. Ezek után egyszerű a problémát megoldani. Gyors és egyszerű magyarázata: Van egy változóm, aminek értéke legyen most 16. A változó legyen "a". a=16. Ha adok egy olyan utasítást hogy a=a+a, akkor azt mit jelentene a számítógépnek? 16=16+16!? De ez lehetetlen!! A számítógép ezt úgy értelmezi, hogy az "a" (új) értékét úgy kapja meg hogy összeadja "a"-t és "a"-t. Azaz ??=16+16. A memóriában mindig a legutolsó (változatlan) változóérték lesz elérhető.

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: Github 2.4 megoldása

Tanulságok, tapasztalatok, magyarázat: Ez egy kicsit trükkös feladat. Először arra gondoltam hogy a kurzot mégis hogyan tudom pattogtatni. Pedig ahogy a valóságban is, itt is fontosabb a "Hol fogom pattogtatni?" csak ez után jön a "Mit fogok pattogtatni?" (Hiába van labdám ha nincs hol játszani vele.) Olyan változókra/értékekre van szükségem ami eldönti hogy a konzolon belül mettől meddig mehet a "labdám".

```
Labdapattogtatás
  Folytatás következik...
```

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: bogoMIPS-BHAX

Megoldás forrása:

A program fő része a következő ciklus.

```
    while (a!=0)
    { a<<=1;
      x++;
```

A ciklus feltétele, a kvöetkezőket addig csinálja amíg 0-át nem kap a memóriában. Ezt úgy éri el, hogy a szót 1-el mindig "arrébb tolja". Majd a számlálóhoz, (ami jelzi hogy hányszor kellett "arrébb tolnia" a szót), hozzáad egyet. A ciklus véget is ér ha megkaptuk a szó hosszát.

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása: Github 2.6 megoldása

A Page-Rank feladata az internetes oldalak rangsorolása. A google volt keresőalgoritmusának alapja, innen indult ki minden. Működése: Kezdetben minden oldalnak ad egy ugyanolyan értéket. Ha az egyik oldal

linkeli a másikat, a másik értékéhez hozzáadódik az egyik értéke. A képlet egyszerű, minnél többen hivatkoznak rá, annál előrébb kerül a rangsorban azaz "jobb lesz" és minél jobbak hivatkoznak rá, annál több pontja lesz. Az alábbi algoritmus 4 Honlapot rangsorol.

Pagerank

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

        void kiir (double tomb[], int db)
        {
            for(short i = 0; i < db; ++i)
            printf("%.2f \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);
        }

        int main()
        {
        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};
        int i,j;
         for(;;)
            {
            for(i = 0; i < 4; ++i)
                {
                    PR[i] = 0.0;
                        for(j = 0; j < 4; ++j)
                        PR[i] += (L[i][j] * PRv[j]);
                }

            if(tavolsag (PR, PRv, 4) < 0.00000001)
            break;

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

```
        kiir (PR, 4);
  return 0;
}
```

Egy mátrixban (vagyis c-ben több dimenziós tömbnek szoktuk hívni) vannak az adatok letárolva, hogy ki kire hivatkozik és hányszor. (L) Az első for ciklusban elkezdjük a számolást, az ezen belüli for ciklusban összehasonlítjuk a mátrix felhasználásával minden oldal "értékét". A for ciklus utáni if-be mengézzük hogy a különbség kisebb-e mint 0.00000001. Ha igen akkor kilépünk a végtelen ciklusból, ha nem akkor a PRv-t feltöltjük a PR elemeivel és számolunk tovább a végtelen ciklusban.

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

```
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))
}
```

1. sor kiszámolja a prímeket n-ig (jelen esetben inkább x-ig, de matematkailag általnosabb az n). 2. sor az egymást követő prímek különbségét számolja ki. (mintha az egyik vektort elcsúsztatnánk a másikon) pl: (Az alsó kötőjelek csak a sorok könyebb megértése miatt vannak.)

3_5_7_11_13

2_3_5__7_11 (-)

1_2_2__4__2

3. sor megnézi hogy melyek az ikeprímek. Ahol az előző kivonás különbsége 2, ott vannak az ikerprímek. 4. sor kiveszi az idx alapján az ikerpríemek első tagját. (pl.: a különbségvektorban a 2. helyen lett 2 a különbség --> az eredeti vektor második elemét veszi ki.) 5. sor ugyanazt csinálja mint az előző, cska másik változóba tárolja el és hozzáad az elemekhez kettőt. 6. sor reciprokokat képez az előző 2 változó értékeiből és összeadja azokat. Végül összeadja és visszaadja az rt1plust2-ben lévő értékeket.

Ezután kirajzoltatjuk az értékeket. A grafikonon látszik ahogy egy felső értékhez/határhoz konvergálnak az összegek.

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

Története: 1991-ben a Parade magazin „Kérdezze Marilynt!" című rovatában hozták először a nagy nyilvánosság elé a problémát Mi a következőz kérdésre a helyes válasz? Ön játékos egy televíziós műsorban, és három ajtó közül kell választania. Az egyik mögött rejlik a főnyeremény, egy autó, a másik kettő mögött pedig nincs semmi. Ön a második ajtót választja. Ekkor a műsorvezető kinyitja a másik két ajtó közül az egyiket, mondjuk a harmadikat, ami üres. Majd felteszi a kérdést, hogy szeretne-e változtatni és esetleg másik ajtót választani? A helyes válasz rá, igen, mert ezzel megduplázza a nyerési esélyeit. De hogyan, és miért? Ezt nem túl egyszerű elmagyarázni (mivel matekmatikai bizonyítások ellenére és kételkednek), így a forráskód és egy példa alapján fogom bemutatni.

```
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 program véletlen eseteket generál. Az első sorban meghatároztuk a kísérletek számát. A "jatekos" vektorban megadjuk mit választ a játékos. (A vektor elemeinek száma ugyanannyi mint a kísérletek száma) Majd egy for ciklussal végignézzük a kísérleteket. Ha eltalálta a játékos kap pontot, ha nem akkor a műsorvezető. Majd kiértékeljük hányszor nyert. Ennek alapján kiértékeljük hogy ha változtatott volna, akkor hányszor nyert volna. Majd a megszámolt értékeket kiíratjuk. A nyerési esély 2/3 ha változtatunk, amíg 1/3 ha nem változtatunk.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: 3.1 megoldás

Tanulságok, tapasztalatok, magyarázat: A feladat az általunk használt 10-es számrendszerből az 1-es számrendszerbe átváltás. A program feladata csak annyi, hogy annyi vonalat húz, mint a természetes szám, amit megdunk neki.

```
#include <iostream>
int main()
{
  int a;
  std::cin >> a;
  std::cout << std::string(a, 'I') << "\n";
  return 0;
};
```

Létrehoztam egy a egész típusú változót. Majd a turing gépeket utánozva valamilyen inputot kértem a értékére. Majd a-szor írtam ki az vonalat. (nem akartma egyest használni mert megtévesztő lehet).

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: 3.4 megoldás

Tanulságok, tapasztalatok, magyarázat: A minta illesztésen alapulva legenerálja a lexikális elemzőt, csak a mintát kell megadni, ami alapján figyeli a begépelt szöveget. Itt olyan programot írunk, ami a begépelt szövegből felismeri és megszámolja a valós számokat.

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.

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);
```

Ha a SIGINT jel kezelése nem volt figyelmen kívűl hagyva, akkor a jelkezelo függvény kezelje, máskülönben figyelmen kívűl legyen hagyva.

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

i keztdőértéke nulla. i végértéke kevesebb mint öt. Veszi i-t megnöveli +1-el a jelenlegi értékéhez képest. Effektívebb, i előző értéke nem marad meg.

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

i kezdőértéke nulla. i végértéke kevesebb mint öt. Veszi i-t lemásolja i-t megnöveli az eredeti i értéket +1-el, de a régi érték másolata megmarad. Nem túl effektív ha nincs szükség i előzőjére.

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

i kezdőértéke nulla. i végértéke kevesebb mint öt. A tömb minden i elemét megváltoztatjuk a jelenlegi i-re, aztán növeljük i-t 1-el.

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

i kezdőértéke nulla. i végértéke kevesebb mint n és ha d tömbre muntató mutató következő eleme egyenlő az s mutató által mutatatott tömb következő elemével.

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

Kiiratunk két egész számot. Mindkettőt az f függvény adja vissza. Első esetben az f függvénynek átadjuk az a változót és az a változó 1-el megnövelt értékét. Másodszor pedig az a változó 1-el megnövelt értékét és a-t adjuk átadjuk az f függvénynek.

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

Két egész számot iratunk ki, az egyik az f függvény által visszaadott szám, az a változót átadjuk az f-nek, a másik pedig az a változó.

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

Két egész számot iratunk ki, az egyik az f függvény által visszaadott szám, az a változó memóriacímét átadjuk az f-nek, a másik pedig az a változó.

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

A jelentésü röviden:

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

Sok prímszám van.

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

Sok iker-prímszám van.

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

Véges sok prímszám van.

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

Véges sok prímszám van.

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:

- ```
  int a;
  ```

  egész
- ```
  int *b = &a;
  ```

  egészre mutató mutató
- ```
  int &r = a;
  ```

  egész referenciája
- ```
  int c[10];
  ```

  egészek tömbje
- ```
  int (&tr)[10] = c;
  ```
  ```

- egészek tömbjének referenciája (nem az első elemé)

```
int *d[5];
```

- egészre mutató mutatók tömbje

```
int *h ();
```

- egészre mutató mutatót visszaadó függvény

```
int *(*l) ();
```

- egészre mutató mutatót visszaadó függvényre mutató mutató

```
int (*v (int c)) (int a, int b)
```

- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény

```
int (*(*z) (int)) (int, int);
```

- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

Mit vezetnek be a programba a következő nevek?

```
int a;
```

```
int *b = &a;
```

```
int &r = a;
```

```
int c[5];
```

```
int (&tr)[5] = c;
```

```
int *d[5];
```

```
int *h ();
```

```
int *(*l) ();
```

```
int (*v (int c)) (int a, int b)
```

```
int (*(*z) (int)) (int, int);
```

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás videó:

Megoldás forrása: Github 4.1 megoldása

A mátrixok m darab sorral és n darab oszloppal rendelkező táblázatok. Akkor beszélünk háromszögmátrixról, ha a négyzetes mátrix (sorok és oszlopok száma megegyezik) főátlója alatt vagy felett csupa 0 elem található. Az alsó háromszögmátrix elemeit sorfolytonosan bejárva el tudjuk helyezni egy tömbben, az így kapott elemek száma n(n+1)/2 lesz. Jelen esetben 3 sorral (a programban nr változó tárolja) fogunk dolgozni, tehát az elemek száma 6. A `malloc` függvény képes lefoglalni a dinamikus területen egy, a paramétereként kapott méretű területet. Ezt követően vagy a lefoglalt terület kezdőcímét, vagy NULL-t ad vissza (hiba esetén). A man 3 malloc parancs terminálba való beírásával megkaphatjuk a függvény pontos szintaktikai leírását. Egyetlen paramétere (size_t size) azt mondja meg, hogy hány bájtot szeretnénk a memóriában lefoglalni. A `free` függvény felszabadítja a lefoglalt területet. Manuáljából (man 3 free) megtudhatjuk, hogy bemeneti paramétere a memóriát azonosító pointer (ptr).

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

int
main ()
{
    int nr = 3;
    double **tm;

    printf("%p\n", &tm);

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

    printf("%p\n", tm);
```

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

    }

    printf("%p\n", tm[0]);

    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");
    }


    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;
}
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: Github 4.2 megoldása

Az EXOR titkosító lényegében a logikai vagyra, azaz a XOR műveletre utal, mely bitenként összeshasonlítja a két operandust, és mindig 1-et ad vissza, kivéve, amikor az összehasonlított 2 Bit megegyezik, mert

akkor nullát. Tehát 2 operandusra van szükségünk, ez jelen esetben a titkosítandó bemenet, és a titkosításhoz használt kulcs. Jó esetben a kettő mérete megyegyezik, így garantálható, hogy szinte feltörhetetlen kódot kapunk, mivel túl sokáig tart annak megfejtése. A mi pédánkban természetesen nem lesz ilyen hosszú a kulcs, mivel ki is szeretnénk próbálni a programot. Viszont ha a kulcs rövidebb, mint a titkosítandó szöveg, akkor a kulcs elkezd ismétlődni, ami így egy biztonsági kockázat.

```
#define MAX_KULCS 100
#define BUFFER_MERET 256
```

A kulcs méret és a buffer méretének maximumát konstansban tároljuk el, ezek nem módosíthatóak. A szintaxisa is másabb, mint egy változó definiálásánál, itt lényegében azt adjuk meg, hogy mivel helyettesítse a megadott nevet a program a forrásban. Nem csak számokat használhatunk konstansként, hanme stringeket is. Érdekessége, hogy nem program futtatásakor történik meg a behelyettesítés, hanem a már a fordítás alatt, tehát a gépi kód már behelyettesített értékeket tartalmazza.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: Github 4.4 megoldása

Tanulságok, tapasztalatok, magyarázat: Mivel ez a feladat a korábban létrehozott EXOR-titkosítónkra épül, ezért dolgozhatunk ugyanazokkal a szövegfájlokkal (`tiszta.txt` fájlban az eredeti szöveg és `titkos.szoveg` állományban a titkosító által generált szöveg). Az előző feladatban egy 8 számjegyű kulccsal dolgoztunk (56789012), ennek megfelelően a `t.c` kódja is ehhez alkalmazkodik (#define KULCS_MERET 8). A `tiszta_lehet` függvényben feltételezzük, hogy a tiszta szöveg valószínűleg tartalmazza a leggyakoribb magyar szavakat pl. hogy, nem, az, ha. Az átlagos szóhossz definiálásával csökkentjük a potenciális töréseket, ennek ellenére valószínűleg nem csak a helyes kulcsot (és tiszta szöveget) fogjuk visszakapni.

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

Ezúttal is meghatározunk bizonyos konstansokat, ebből a kulcs_meret érdekes, mert feltételezzük, hogy a kulcs 8 elemből áll, már itt látni, hogy ez nem lenne túl hatékony a való életben.

```
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;
}
```

Az `atlagos_szohossz` függvénnyel kiszámítjuk a bemenet átlagos szóhosszát, argumentumként átadjuk egy tömböt, és annak a méretét. Majd egy `for` ciklussal bejárjuk, és minden elem után hozzáadunk 1-et az `sz` változóhoz. Visszatérési értékként pedig a tömb méretének és a számlálónka a hányadosát adjuk.

```
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");

}
```

4.5. Neurális OR, AND és EXOR kapu

R

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

Megoldás forrása: Github 4.5 megoldása

Az R nyelv neurális hálózatot fogunk létrehozni, mely képes "tanulni", és megközelíteni az átalunk megadott megfelelő értékeket. A hálózat a nevét a neuronról kapta, mely agyunk egy sejtje. Feladata az elektromos jelek összegyűjtése, feldolgozás és szétterjesztése. Az a feltételezés, hogy az agyunk információfeldolgozási képességét ezen sejtek hálózata adja. Éppen emiatt a mesterséges intelligencia kutatások során ennek a szimulálást tűzték ki célul.

Az első részben a logikai vagyot tanítjuk meg a neurális hálózatnak, mely 1-et ad vissza, kivéve, ha mind a két operandusa 0, mert akkor 0-át.

```
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])
```

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: Github 4.6 megoldása

Ebben a feladatban folytatjuk a elmélkedést a neuron hálózatokra, ezen belül perceptronokról lesz szó. Ez egy algoritmus amely a számítógépnek "megtanítja" a bináris osztályozást. Ide is berakhatnám az előző fejezetben lévő képet a neuronról, ami egy bemenetet kap, és egy bizonyos pontot elérve "tüzel", ad egy kimenetet. Itt is hasonló dologról van szó:

Tehát van egy halmaz amiben vannak piros és fekete pontok, a fekete pontok a vonal felett vannak, a pirosak pedig alatta. Adok a perceptronnak bemenetként egyet-egyet mind a kettőből, és a képes lesz megmondani a többite, hogy a vonal felett van-e, vagy alatta. Ezért nevezzük bináris osztáylozásnak, mert van a vonal felettiek osztálya és az alattiaké, ez a kapcsolat könnyen reprezentálható eggyel és nullával.

Akkor lássuk magát a programot. Most C++-ban fogunk dolgozni, melyről tettem már említést a Java-s feladatban, szóval annyit már tudsz, hogy ez is egy objektum-orientált nyelv, szóval a class-okat képtelenség lesz elkerülni. 3 fájlre lesz szükség, abból az egyikkel, `mandelpng.cpp`-vel most nem foglalkoznék, hiszen a következő fejezetben pont erről lesz szó. Egyelőre legyen elég annyi, hogy ezzel tudunk készíteni egy képet, ami a Mandelbrot halmazt ábrázolja.

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...

IV. rész

Irodalomjegyzék

10.3. Általános

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

10.4. C

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

10.5. C++

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

10.6. 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.