

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. február 16, v. 0.0.3

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>

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Radi, Daniel	2019. március 12.	

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

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	9
2.6. Helló, Google!	10
2.7. 100 éves a Brun tétel	11
2.8. A Monty Hall probléma	11
3. Helló, Chomsky!	13
3.1. Decimálisból unárisba átváltó Turing gép	13
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	13
3.3. Hivatkozási nyelv	14
3.4. Saját lexikális elemző	14
3.5. l33t.1	15
3.6. A források olvasása	15
3.7. Logikus	16
3.8. Deklaráció	17

4. Helló, Caesar!	18
4.1. int *** háromszögmátrix	18
4.2. C EXOR titkosító	18
4.3. Java EXOR titkosító	18
4.4. C EXOR törő	18
4.5. Neurális OR, AND és EXOR kapu	19
4.6. Hiba-visszaterjesztéses perceptron	19
5. Helló, Mandelbrot!	20
5.1. A Mandelbrot halmaz	20
5.2. A Mandelbrot halmaz a std::complex osztállyal	20
5.3. Biomorfok	20
5.4. A Mandelbrot halmaz CUDA megvalósítása	20
5.5. Mandelbrot nagyító és utazó C++ nyelven	20
5.6. Mandelbrot nagyító és utazó Java nyelven	21
6. Helló, Welch!	22
6.1. Első osztályom	22
6.2. LZW	22
6.3. Fabejárás	22
6.4. Tag a gyökér	22
6.5. Mutató a gyökér	23
6.6. Mozgató szemantika	23
7. Helló, Conway!	24
7.1. Hangyaszimulációk	24
7.2. Java életjáték	24
7.3. Qt C++ életjáték	24
7.4. BrainB Benchmark	25
8. Helló, Schwarzenegger!	26
8.1. Szoftmax Pysssss MNIST	26
8.2. Szoftmax R MNIST	26
8.3. Mély MNIST	26
8.4. Deep dream	26
8.5. Robotpszichológia	27

9. Helló, Chaitin!	28
9.1. Iteratív és rekurzív faktoriális Lisp-ben	28
9.2. Weizenbaum Eliza programja	28
9.3. Gimp Scheme Script-fu: króm effekt	28
9.4. Gimp Scheme Script-fu: név mandala	28
9.5. Lambda	29
9.6. Omega	29
 III. Második felvonás	 30
10. Helló, Arroway!	32
10.1. A BPP algoritmus Java megvalósítása	32
10.2. Java osztályok a Pi-ben	32
 IV. Irodalomjegyzék	 33
10.3. Általános	34
10.4. C	34
10.5. C++	34
10.6. Lisp	34

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 alkálni 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 (hogyan lássuk mást is) példával.

Hogyan nyomjuk?

Ránts 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 dlatex 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
dlatex 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 'dlatex' 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 találsz 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

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!

Ha egy végtelen ciklust futtatunk a mag 100% terhelt lesz, ugyanis végtelenszer ismétli önmagát.

```
#include <omp.h>

int main()
{
    #pragma omp single
    {for(;;){}}
}
```

Ha egy feladat megáll vagy elaltatjuk a sleep paranccsal, a mag 0% terhelt lesz.

```
#include <unistd.h>

int main()
{
    for(;;){usleep ( 100000 );}
}
```

Ha egy végtelent ciklust paralell futtatunk minden magon, minden mag 100% terhelt lesz. -openmp szükséges hozzá

```
#include <omp.h>

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

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;
    }

    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 építő `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.

Tegyük fel, hogy létezik olyan program (T100), ami el tudja dönteni egy másik programról, hogy van-e benne végtelen ciklus. Ha van, akkor megáll a program, ha nincs, akkor pedig végtelen ciklusba kezd. Létrehozunk egy új programot (T1000) az előzőt (T100) felhasználva és ha a T100 megállt, akkor végtelen ciklusba kezd, ha pedig a T100 kezdett végtelen ciklusba, akkor megáll.

A megállás csak akkor lehetséges, ha a T100 nem áll meg, de ez pedig csak akkor lehet, ha a második argumentumként kapott saját programunk megáll.

Ebből ellentmondásra jutottunk, tehát nem lehet ilyen programot írni.

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: [Valtozo felcserelesek](#)

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int a = 1;
```



```
int b = 2;
string sel(" ");
cout << "A kezdő számok: " << a << sel << b << "\n";

int c = a;
a = b;
b = c;
cout << "Új változós felcserelessel: " << a << sel << b << "\n";

a = a+b;
b = a-b;
a = a-b;
cout << "Visszacseréles új változó nélkül: " << a << sel << b << "\n";
}
```

Új változóval: Bevezetünk egy új változót amire "elmentjük" az 'a' változón tárolt információt. Ekkor 'a' változót felülírhatjuk 'b' változóval majd a 'b' változót az új változóval.

Új változó nélkül: Összeadjuk a két változót és megkapjuk az összegüket az egyik változón(itt 'a'). A másik változón(itt 'b') kivonjuk az összegből('a') a 'b' változót, a maradék 'a' eredeti értéke lesz. Ez után az összeget tartalmazó változón('a') kivonuk önmagából a 'b' értéket, ami jelenleg 'a' eredeti értéke, így megkapjuk a 'b' eredeti értéket.

2.4. Labdapattogás

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

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

Megoldás forrása: [Labdapattogtatás](#)

Megjegyzés: Minden esetben felvesszük a kódhoz szükséges headert, amit a forrásban meg lehet találni.

Elkészítjük az ablakot a terminalból és felvesszük az adatait.

```
WINDOW *ablak;
ablak = initscr ();
```

Egy végtelen for ciklusba fog a program futni, ugyanis most nem lesz szükség a labda megállítására.

Felvesszünk két változót (a for cikluson kívül) az ablak (x,y) méretének tárolására majd belementjük az értéket minden ciklusban. Ezt a forrásban megfigyeztük hogy csak az ablak feléig dolgozzunk.

```
getmaxyx ( ablak, amy , amx );
```

Egyszerű forciklussal megrajzoljuk a határértékeket tetszőleges karakterekkel, illetve a két maxértékkel a legtávolabbi pontot, az ellentéző sarkot. Ennek csak esztétikai jelentősége van.

```
for(int a = 0;a<amx;a++){mvprintw ( amy, a, "=" );}  
for(int b = 0;b<amy;b++){mvprintw ( b, amx, "|" );}  
mvprintw ( amy, amx, "/" );
```

Felveszünk két új változót a labda X,Y értékeinek tárolására a végtelen cikluson kívül. Ez a labda pozíció kezdő értéke. Felrajzoljuk a labdát a jelenlegi pozíciójára majd frissítjük az ablakot, várunk hogy a mozgás emberi szemmel is észlelhető legyen majd eltöröljük az ablakon megjelenített elemeket.

```
mvprintw ( y, x, "O" );  
refresh ();  
usleep (100000);  
clear();
```

Felveszünk két új változót a labda mozgási sebessége meghatározására és hozzáadjuk az értéket az előző pozícióhoz.

```
x = x + xmo;  
y = y + ymo;
```

Egyszerű if függvénnyel ellenőrizzuk hogy elérte-e a labda a határértéket. Ha elérte a sebességet reciprokára állítjuk, ezzel a labda mozgását az ellentéző irányba megfordítva.

```
if ( x>=amx-1 ) {xmo = xmo * -1;}  
if ( x<=0 ) {xmo = xmo * -1;}  
if ( y<=0 ) {ymo = ymo * -1;}  
if ( y>=amy-1 ) {ymo = ymo * -1;}
```

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás forrása:

```
#include <stdio.h>  
int  
main (void)  
{  
    int h = 0;  
    int n = 0x01;  
    do  
        ++h;  
    while (n <= 1);  
    printf ("A szóhossz ezen a gépen: %d bites\n", h);  
    return 0;
```

```
}  
}
```

Az egy érteku változot a bitshift operátorral eltoljuk amig lehet és minden lépést megszámlolunk. Az eredmény az érteku bit árteku mérete. Ez a változó típusától is függ.

2.6. Helló, Google!

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

Megoldás forrása: [Pagerank](#)

Az alábbi funkció kiírja a belevezettet értékeket érkezesi sorrend szerint százados helyiértékig.

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

Az alábbi funkció kiszámolja a PageRank és a határérték távolságát a négy oldal jóságának felhasználásával, majd visszaadja azt double értékként.

```
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);  
}
```

A main-ben felvesszük az oldalak linkjeinek számát mátrixként, illetve a pagerankot és a határértéket vektorként.

Egy for függvényben elkezdjük a számolást. Ezen belül dupla for függvénnyel összehasonlítjuk a mátrix minden koordinátájának felhasználásával a page "jóságot".

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

Amennyiben a különbség kisebb mint 0.00000001 akkor kilépünk a végtelen ciklusból, ha nem PRv-t feltöltjük PR elemeivel. Ha ciklus véget ér, kiíratjuk az értékeket.

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

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
  rtlplust2 = 1/t1primes+1/t2primes
  return(sum(rtlplust2))
}
```

1. sor kiszámolja a prímeket x-ig.
2. sor az egymást követő prímelek különbségét veszi. 3-2, 5-3, 7-5, 11-7, 13-11.
3. sor megnézi hol 2 a különbség. Ahol igaz, azok ikerprim párok.
4. sor a pár első tagját eltárolja, ahol 3 teljesül.
5. sor a második tagot az első+2 vel képzi és tárolja, ahol 4 teljesül.

Végül reciprokokat képez, összeadja őket majd visszaadja az értéket.

A forrás további részében kirajzoltatjuk és láthatjuk, hogy egy felső 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

```
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. Meghatározzuk a kísérletek számát majd a játékos vektorban véletlenszerűen megadjuk mit választ esetenként. A vektorok mérete a kísérletek számával egyenlő.

Egy for ciklussal végigmegyünk a kísérletek száman és megnézzuk hogy eltalálta-e az eredményt.

Ha nem találta el, a műsorvezető kap pontot.

Ha nyer, kiértékeljük hogy hányszor nyert volna változtatás nélkül. Egyéb esetekben változtatással nyert volna.

Megszámoljuk a vektorokon az eseményeket majd kiíratjuk az eredményeket..

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása:

```
#include <iostream>
#include <string>
using namespace std;
int
main ( void )
{
    int num;
    string output;
    cin >> num;
    for(int i=0;i<num;++i) {output.append("1");}
    cout << output << "\n";
}
```

Unáris szám annyi 1-es karakterből áll, ahány az értéke. Megadunk egy int értéket majd létrehozunk egy üres kimeneti stringet. A for függvény annyiszor fut le ahány a bementi számunk. Minden esetben hozzáad(appendál) egy karaktert a stringhez, ami egy 1-es karakter. Végül kiiratjuk a stringet.

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

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

$S \rightarrow abclA$

$A \rightarrow aABclabc$

$cB \rightarrow Bc$

$bB \rightarrow bb$

$S \rightarrow aSBC \mid aBC$ $CB \rightarrow BC$ $aB \rightarrow ab$ $bB \rightarrow bb$ $bC \rightarrow bc$ $cC \rightarrow cc$

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:

```
long long int
```

Új adattípus a C99 nyelvben. Explicit boolean adattípus komplex számokhoz. 32 bit méret, [-2,147,483,647, +2,147,483,647] közötti értékkel.

Aritmetikai és vezérlő utasítások függvényen belül, blokkban helyezkednek el. A tevékenység pontos megfogalmazása amit nem lehet részletezni, pontos jelentéssel bír.

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!

Felvesszük a megoldásunkat tároló intet 0-ként, hiszen ehhez adunk majd hozzá.

```
%{  
#include <stdio.h>  
int realnumbers = 0;  
%}
```

Definiáljuk mi is az amit keresünk.

```
digit [0-9]
```

Valahányszor valós számot találunk, a (megoldás) darabszámot növeljük eggyel, és a kimeneti stringen a valós számot kiemeli egy "realnum=" -al.

```
        {digit}*(\.{digit}+)? {++realnumbers;
printf("[realnum=%s %f]", yytext, atof(yytext));}
```

Lehívjuk az yylex-et. A szokásos módon kiírjuk az eredményt standard outputon.

```
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", realnumbers);
    return 0;
}
```

3.5. l33t.l

Lexelj össze egy l33t ciphert!

```
%{
#include <stdio.h>
%}
leet [e]
%%
{leet} {printf("3");}
%%
int
main ()
{
    yylex ();
    return 0;
}
int yywrap() {return(1);}
```

A program valahányszor az "e" karaktert olvass a bemeneten, helyette egy "3"-as karaktert ír ki. Semmilyen más karakter nem változik. A main utáni rész megegyezik az előző felattal.

3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezelő)==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 jelkezelő függvény kezelje. (Miótan 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 megyránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

- i. Ha a SIGINT jel kezelését végrehajtjuk, akkor a jel kezelését a 'jelkezelő' végezze.

```
if(signal(SIGINT, SIG_IGN) != SIG_IGN)
    signal(SIGINT, jelkezelő);
```

- ii. Ötször lefutó for ciklus.

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

- iii. Ugyan az mint az előző.

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

- iv.

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

A 'tomb' elemeinek az index értékét adja meg.

- v. Egy for ciklus ami addig fut le amíg eléri n-t és d indexértékét s indexértékéhez hasonlítja.

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

- vi. Két számot váró kiírás két függvényből.

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

- vii. Két számot váró kiírás egy függvényből és egy változóból.

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

- viii. Két számot váró kiírás egy nem nulla függvényből és egy változóból.

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

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

1. minden x re van egy annál nagyobb prímszám.
2. minden x re van egy annál nagyobb ikerprím számpár.
3. Van olyan y aminél minden x prím kisebb.
4. Van olyan y aminél bármely nagyobb szám prímszám.

3.8. Deklaráció

- egész

```
int a;
```

- egészre mutató mutató

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

- egész referenciája

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

- egészek tömbje

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

- egészek tömbjének referenciája (nem az első elemé)

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

- egészre mutató mutatók tömbje

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

- egészre mutató mutatót visszaadó függvény

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

- egészre mutató mutatót visszaadó függvényre mutató mutató

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

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

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

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

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

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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:

Tanulságok, tapasztalatok, magyarázat...

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:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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 iteráció bejárta z_n komplex számokat!

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

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 elspájzold 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...

DRAFT

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Pysssss MNIST

aa Python a_2^2 aa a_2 aa

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...

DRAFT

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:

DRAFT

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.

DRAFT

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

DRAFT

10.3. Általános

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

10.4. C

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

10.5. C++

[BMECPP] Benedek, Zoltán & 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.