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	Róka, Dávid	2019. március 20.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Egy magot 100%-ban a while illetve a for ciklus alkalmazásával is pörgethetünk.

Megoldás while ciklussal:https://github.com/rokadavid99/Prog1/blob/master/loopw.c

```
#include <stdio.h>
int main(){

    while(1);

    return 0;
}
```

A két megoldás közül ez az egyszerűbb, mondhatni stílusosabb.

Megoldás for ciklussal:https://github.com/rokadavid99/Prog1/blob/master/loopf.c

```
#include <stdio.h>


int main(){
    for(;;);

    return 0;


}
```

Viszont ez a megoldás egyértelművé teszi a program megértését a két db ";;" jel miatt, ami kifejezi a szándékunkat.

Több mag 100%-on:https://github.com/rokadavid99/Prog1/blob/master/allcore100.c

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

int main(){

#pragma omp parallel
#pragma omp while

  while(1);

return 0;
}
```

Ahhoz, hogy több magot 100%-on tudjunk futtatni, a parallel pragma-t kell fehasználnunk, ami az őt követő utasításokat több szálon futtatja.Nem szabad figyelmen kívűl hagyni, hogy a kód elején meg kell hívnunk az omp.h headert.

Egy mag 0%-on:https://github.com/rokadavid99/Prog1/blob/master/loop0.c

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


int main(){

    while(1){

    sleep(100);

     }
    return 0;

}
```

Amikor egy magot szeretnénk használni és azt is 0%-on, akkor azt a sleep függvény segítségével tehetjük meg. A sleep függvény használatához meg kell hívni az unistd.h headert.

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

```
  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

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

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

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

Valtozók cseréje összeadással/kivonással:

```
#include<stdio.h>

int
main()
{
    int a,b;
    a=6;
    b=13;

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

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

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


    return 0;
}
```

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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.)

Labdapattogtatás if használatával:

```
#include <curses.h>
#include <unistd.h>
int main ( void )
{
    WINDOW *ablak;
    ablak = initscr ();
    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;
    int mx;
    int my;
    for ( ;; ) {


        getmaxyx ( ablak, my , mx );

        mvprintw ( y, x, "O" );
        refresh ();

         usleep ( 50000 );
  clear();

        x = x + xnov;
        y = y + ynov;
        if ( x>=mx-1 ) {
            xnov = xnov * -1;
        }
        if ( x<=0 ) {
            xnov = xnov * -1;
        }
        if ( y<=0 ) {
            ynov = ynov * -1;
        }
        if ( y>=my-1 ) {
            ynov = ynov * -1;
        }
```

```
    }
    return 0;
}
```

Labdapattogtatás if nélkül: https://github.com/rokadavid99/Prog1/blob/master/labda.c

```
#include <curses.h>
#include <unistd.h>
#include <stdlib.h>
int main ( void )
{
    WINDOW *ablak;
    ablak = initscr ();
    int mx;
    int my;
    int xj = 0, xk = 0, yj = 0, yk = 0;
    for ( ;; ) {
        getmaxyx ( ablak, my , mx );


        xj = (xj - 1) % mx;
        xk = (xk + 1) % mx;
        yj = (yj - 1) % my;
        yk = (yk + 1) % my;


        mvprintw ( abs (yj + (my - yk) ), abs (xj + (mx - xk)) , "O" );
        refresh ();
        usleep ( 50000 );
  clear();
    }
    return 0;
}
```

A feladat megoldható a maradékos osztás művelet használatával(%) hiszen az előző példából tudhatjuk hogy a labda mozgatása során pl az oszlop kordinátája így változik 1,2,3,4,5,4,3,2,1,2... tehát folyamatosan hullámzik ameddig el nem éri a határértéket ez maradékos osztással is elérhető de mivel az az elért érték után rögtön nullára esik vissza: 1,2,3,4,5,0,1,2,3,4,5,0... ezért 2 sorozat öszemosására van szükségünk:

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: https://github.com/rokadavid99/Prog1/blob/master/szohossz.cpp

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

```
int main()
{

int szo=1;
int lepes=1;

while(szo<<=1)
 ++lepes;

printf("%i\n",lepes);

}
```

Ez a bitshift operátor, a megadott változó memoriában tárolt bitjein végez műveletet, eltolja azokat annyival (itt balra) amekkora értéket megadunk neki.A képernyőn megjelenő szám a 32, tehát ennyi bitből áll egy gépi szó.

2.6. Helló, Google!

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

Megoldás forrása: https://github.com/rokadavid99/Prog1/blob/master/pagerank.c

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

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

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

  }

double tavolsag(double pagerank[],double pagerank_temp[],int db)
{
double tav = 0.0;
int i;
for(i=0;i<db;i++)
tav +=abs(pagerank[i] - pagerank_temp[i]);
return tav;
}

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};

  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) < 0.0000001)
    break;

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

  }
  kiir (PR,4);
  return 0;

}
```

A PageRank egy módszer weblapok osztályozására, az alapötlet: azok a lapok jobb minőségüek amelyekre jobb minőségű lapok mutatnak.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

```
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")
```

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

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat...

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása:https://github.com/rokadavid99/Prog1/blob/master/turing.c

```
#include <stdio.h>

int
main()
{
  int a, db=0;
  printf("Adj meg egy decimalis szamot!\n");
  scanf("%d", &a);
  printf("Atvaltva unarisba:\n");
  for (int i = 0; i < a; i++)
  {
    printf("|");
    db++;
    if (db % 5 == 0)
                {
                printf(" ");
                }
  }
  printf("\n");
  return 0;
}
```

Jó ha tisztázzuk, a feladatban szereplő fogalmakat.A decimális számrendszer a tizes számrendszert jelenti, középsuliban ebben számoltunk a legtöbbet. Az unáris számrendszer pedig az egyes számrendszert jelenti, ami talán a legegyszerűbb az összes számrendszer közül. Lényege, hogy a természetes számokat megfelelő mennyiségű szimbólummal ábrázoljuk, esetünkben ez a "|".

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

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

A generatív grammatika fogalma Noam Chomsky, amerikai nyelvész professzor nevéhez fűződik. Négy alkotóeleme van a generatív grammatikának: terminális szimbólumok, nem terminális jelek, kezdőszimbólum ill. helyettesítési szabályok. A grammatikákat osztályozással különíthetjük el egymástól. Ezek szerint lehet: általános (0-s típusú), környezetfüggő (1-es típusú), környezetffügetlen (2-es típusú) és reguláris (3-as típusú). A feladat szövegében a környezetfüggő (1-es típusú) osztály van említve. Itt a képzési szabály bal, illetve jobb oldalán is szerepelhetnek terminális szimbólumok. (környezetfüggetlen grammatikák esetén a bal oldalon csak nemterminális szimbólum lehet)

```
S, X, Y "változók" - ezek nemterminálisok
a, b, c "konstansok" - ezek terminálisok
S -> abc, S -> aXbc, Xb -> bX, Xc -> Ybcc, bY -> Yb, aY -> aaX, aY -> aa -  ↩
   ezek a képzési szabályok
S lesz a kezdőszimbólum
Ha jól alkalmazzuk a képzési szabályainkat, akkor ezt kapjuk:

                S (S -> aXbc)
                aXbc (Xb -> bX)
                abXc (Xc -> Ybcc)
                abYbcc (bY -> Yb)
                aYbbcc (aY - aa)
                aabbcc

-----------------------------------------------------------------
A nyilak segítségével tudjuk jelölni a képzési szabályt. (miből -> mi lesz)
Ugyanezen nyelv egy másik reprezentációja:

                S (S -> aXbc)
                aXbc (Xb -> bX)
                abXc (Xc -> Ybcc)
                abYbcc (bY -> Yb)
                aYbbcc (aY -> aaX)
                aaXbbcc (Xb -> bX)
                aabXbcc (Xb -> bX)
                aabbXcc (Xc -> Ybcc)
                aabbYbccc (bY -> Yb)
                aabYbbccc (bY -> Yb)
                aaYbbbccc (aY -> aa)
                aaabbbccc
```

Onnan tudhatjuk, hogy ez a grammatika környezetfüggő, hogy a képzési szabályokban van olyan, ahol a nyíl bal oldalán is láthatunk terminális szimbólumot. Fentebb már említettem, most nyomatékosítom, hogy ez egy környezetfüggetlen nyelvben nem létezik.

Másik környezetfüggő grammatika:

```
A, B, C "változók" - ezek nemterminálisok
a, b, c "konstansok" - ezek terminálisok
```

```
A -> aAB, A -> aC, CB -> bCc, cB -> Bc, C -> bc - ezek a képzési szabályok
A lesz a kezdőszimbólum
Ha jól alkalmazzuk a képzési szabályainkat, akkor ezt kapjuk:

                A (A -> aAB)
                aAB (A -> aC)
                aaCB (CB -> bCc)
                aabCc (C -> bc)
                aabbcc

-----------------------------------------------------------------
Ugyanezen nyelv egy másik reprezentációja:

                A (A -> aAB)
                aAB (A -> aAB)
                aaABB (A -> aAB)
                aaaABBB (A -> aC)
                aaaaCBBB (CB -> bCc)
                aaaabCcBB (cB -> Bc)
                aaaabCBcB (cB -> Bc)
                aaaabCBBc (CB -> bCc)
                aaaabbCcBc (cB -> Bc)
                aaaabbCBcc (CB -> bCc)
                aaaabbbCccc (C -> bc)
                aaaabbbbcccc
```

Nagyban hasonlít a két nyelvtan, csak a változókat és a képzési szabályokat írtam át, mégis egy másik környezetfüggő generatív nyelvet kaptunk.

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.

Először is hogy megértsük meg kell adnunk a származtatási szabályok halamzát:

```
<utasítás> ::= <kifejezés> | <összetett_utasítás> | <feltételes_utasítás> | ↩
    <while_utasítás> | <do_utasítás> | <for_utasítás> | <switch_utasítás> | ↩
    <break_utasítás> | <continue_utasítás> | <return_utasítás> | < ↩
   goto_utasítás> | <cimke_utasítás> | <nulla_utasítás>

<kifejezés> ::= <értékadás> | <függvényhívás>
<értékadás> ::= <változó><szám>
<változó> ::= <betű>{<betű>}
<betű> ::= a-z
<szám> ::= <számjegy>{<számjegy>}
<számjegy> ::= 0|1|2|3|4|5|6|7|8|9
```

```
<függvényhívás> ::= <típus><függvénynév>
<típus> ::= <betű>{<betű>}
<függvénynév> ::= <betű>{<betű>}

<összetett_utasítás> ::= <deklarációlista> | <utasításlista>
<deklarációlista> ::= <deklaráció>{<deklaráció>}
<deklaráció> ::= <típus><változó>
<utasításlista> ::= <utasítás>{<utasítás>}

<feltételes_utasítás> ::= if<kifejezés><utasítás> | if<kifejezés><utasítás> ↩
   else<utasítás>

<break_utasítás> ::= break

<while_utasítás> ::= while<kifejezés><utasítás> | while<kifejezés><utasítás ↩
   ><break_utasítás>

<do_utasítás> ::= do<utasítás>while<kifejezés> | do<utasítás>while< ↩
   kifejezés><break_utasítás>

<for_utasítás> ::= for([<kifejezés>][<kifejezés>][<kifejezés>])<utasítás> | ↩
    for([<kifejezés>][<kifejezés>][<kifejezés>])<utasítás><break_utasítás>

<switch_utasítás> ::= switch<kifejezés><utasítás> | switch<kifejezés>< ↩
   utasítás><case><kifejezés><default> | switch<kifejezés><utasítás>< ↩
   break_utasítás>

<continue_utasítás> ::= continue

<return_utasítás> ::= return | return<kifejezés>

<goto_utasítás> ::= goto<azonosító>

<cimke_utasítás> ::= <azonosító>
<azonosító> ::= <cimke>
<cimke> ::= <betű>{<betű>}

<nulla_utasítás> ::=
```

A BNF a rövidítése a Backus-Naur formának, ami egy szintaxis a környezet-független nyelvtanok leírására. Működéséhez szükség van a származtatási szabályok halmazára. Ezt az előbb adtam meg.

```
#include<stdio.h>

int main()
{
    int y;
    for (int a = 0; a < 10; a++)
    {
        y=a*2;
```

```
        printf("%d \n", y);
    }

    return 0;
}
```

Megoldás forrása:

Láthatjuk, hogy a C89-es nem tudja lefordítani azokat a kódot, ahol a for ciklus kezdetén deklarálni akarunk. Tehát, a fordítónk hibát jelzett, viszont megmondta, hogy milyen fordítót használjunk, ha azt szeretnénk, hogy a program működjön.("loop initial declarations are only allowed in C99 or C11 mode")

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!

A Flex segítségével lehet, egy megadott kód alapján lexikális elemzőt generálni C-ben. Tehát, magát a C programot nem mi fogjuk megírni, hanem a lexer. Az ".l" kiterjesztésű fájlok fordítása más, mint például a ".c"-jé. Először az alábbi sorra van szükségünk:

```
lex -o realnumber.c realnumber.l
```

Majd, a már jól ismert gcc jön egy kis kiegészítéssel, amire a linkelés miatt van szükség:

```
gcc realnumber.c -o realnumber -lfl
```

Maga a kód pedig így néz ki:

```
%{
#include <stdio.h>
int realnumbers = 0;
%}
digit [0-9]
%%
{digit}*(\.{digit}+)? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
 yylex ();
 printf("The number of real numbers is %d\n", realnumbers);
 return 0;
}
```

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.5. l33t.l

Lexelj össze egy l33t ciphert!

A leet(l33t vagy 1337) egy olyan eljárás, amiben az írott szöveg betűit, főleg számokkal, de akár más ASCII karakterekkel is helyettesítik. A könnyebb megértés érdekében itt van egy példa.

```
%{
  #include <stdio.h>
  #include <stdlib.h>
  #include <time.h>
  #include <ctype.h>

  #define L337SIZE (sizeof l1337d1c7 / sizeof (struct cipher))

  struct cipher {
    char c;
    char *leet[4];
  } l1337d1c7 [] = {

  {'a', {"4", "4", "@", "/-\\"}},
  {'b', {"b", "8", "|3", "|}"}},
  {'c', {"c", "(", "<", "{"}},
  {'d', {"d", "|)", "|]", "|}"}},
  {'e', {"3", "3", "3", "3"}},
  {'f', {"f", "|=", "ph", "|#"}},
  {'g', {"g", "6", "[", "[+"}},
  {'h', {"h", "4", "|-|", "[-]"}},
  {'i', {"1", "1", "|", "!"}},
  {'j', {"j", "7", "_|", "_/"}},
  {'k', {"k", "|<", "1<", "|{"}},
  {'l', {"l", "1", "|", "|_"}},
  {'m', {"m", "44", "(V)", "|\\/|"}},
  {'n', {"n", "|\\|", "/\\/", "/V"}},
  {'o', {"0", "0", "()", "[]"}},
  {'p', {"p", "/o", "|D", "|o"}},
  {'q', {"q", "9", "O_", "(,)"}},
  {'r', {"r", "12", "12", "|2"}},
  {'s', {"s", "5", "$", "$"}},
  {'t', {"t", "7", "7", "'|'"}},
  {'u', {"u", "|_|", "(_)", "[_]"}},
  {'v', {"v", "\\/", "\\/", "\\/"}},
  {'w', {"w", "VV", "\\/\\/", "(/\\)"}},
  {'x', {"x", "%", ")(", ")("}},
  {'y', {"y", "", "", ""}},
```

```
  {'z', {"z", "2", "7_", ">_"}},

  {'0', {"D", "0", "D", "0"}},
  {'1', {"I", "I", "L", "L"}},
  {'2', {"Z", "Z", "Z", "e"}},
  {'3', {"E", "E", "E", "E"}},
  {'4', {"h", "h", "A", "A"}},
  {'5', {"S", "S", "S", "S"}},
  {'6', {"b", "b", "G", "G"}},
  {'7', {"T", "T", "j", "j"}},
  {'8', {"X", "X", "X", "X"}},
  {'9', {"g", "g", "j", "j"}}

// https://simple.wikipedia.org/wiki/Leet
  };

%}
%%
. {

    int found = 0;
    for(int i=0; i<L337SIZE; ++i)
    {

      if(l337d1c7[i].c == tolower(*yytext))
      {

        int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0));

          if(r<91)
          printf("%s", l337d1c7[i].leet[0]);
          else if(r<95)
          printf("%s", l337d1c7[i].leet[1]);
        else if(r<98)
          printf("%s", l337d1c7[i].leet[2]);
        else
          printf("%s", l337d1c7[i].leet[3]);

        found = 1;
        break;
      }

    }

    if(!found)
       printf("%c", *yytext);

  }
%%
int
```

```
main()
{
  srand(time(NULL)+getpid());
  yylex();
  return 0;
}
```

Megoldás forrása:

A kód elején lévő `#define`-os rész arra szolgál, hogy ha hivatkozunk a `L337SIZE`-ra, akkor azt helyettesíteni fogja a mellette lévő értékkel. Majd létrehozunk egy struktúrát, ami alapján deklaráljuk a `l337d1c7 []` tömböt. Ez a tömb kulcsszerepet fog játszani a programunkban, ugyanis ez tartalmazza a karaktereket és a hozzájuk tartozó helyettesítő karaktereket is. (alatta láthatjuk, hogy mik ezek) A `tolower()` függvény a nagybetűket kicsivé alakítja át. Majd generálunk egy random számot 1 és 100 között. Az if-es részben vizsgáljuk, hogy ha a kapott random szám kisebb, mint 91 akkor a `char *leet[4]` tömb első elemét írjuk ki, ha nem akkor meg haladunk tovább az else ágakon a kód szerint. A `found` arra szolgál, hogy megnézze, hogy a beolvasott karakter benne van-e tömbünkben. A program legvégén érkezünk el csak a main()-hez. Itt a(z) `yylex()` függvénnyel indítjuk el a lexikális elemzést.

3.6. A források olvasása

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

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

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

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

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

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

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

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

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

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

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

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

Megoldás forrása:https://github.com/rokadavid99/Prog1/blob/master/signal.c

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

Ehhez a feladathoz szükségünk van a LaTex-re, ami egy igen hasznos, ám kevésbé elterjedt szövegszerkesztő. A feladatban szereplő parancsok, sokak számára furcsának tűnhetnek elsőre, de nem kell megijedni, mert a mi drága szövegszerkesztőnk ezekből a parancsokból értelmes matematikai kifejezéseket csinál. Sőt, amikor futtatunk egy ilyen szerkesztőt (pl.: TeXworks), akkor egy pdf fájlt is kapunk, ahol szépen olvasható módon látjuk az eredményt.

```
\documentclass{article}
\usepackage[magyar]{babel}
\usepackage[utf8]{inputenc}
\usepackage{amsmath}

\begin{document}

$$1.$$
$(\forall x\exists y((x<y)\wedge(y\textbf{ prím})))$
A prímszámok száma végtelen.
$$2.$$
```

```
$(\forall x\exists y((x<y)\wedge(y\textbf{ prím})\wedge(SSy\textbf{ prím}))
   )$
Az ikerprímek száma végtelen.
$$3.$$
$(\exists y\forall x(x\textbf{ prím})\supset(x<y))$
A prímszámok száma véges.
$$4.$$
$(\exists y\forall x(y<x)\supset\neg(x\textbf{ prím}))$
A prímszámok száma végtelen.



\end{document}
```

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);`

```
int main(){
    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);

    return 0;
}
```

Megoldás forrása:https://github.com/rokadavid99/Prog1/blob/master/deklaracio.c

Ennél a feladatnál arra kell vigyázni, hogy a referenciaérték C++ "feature", amit a gcc fordító nem tud értelmezni, ezért a gcc fordító helyett g++-t kell használnunk.

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Mielőtt nekiugranánk a feladatnak, tisztáznunk kell, hogy mi is az a háromszögmátrix. A háromszögmátrix olyan kvadratikus, azaz négyzetes mátrix (kvadratikus mátrix: olyan mátrix, amelyben a sorainak és oszlopainak a száma megegyezik), melynek a főátlója alatt, vagy felett csupa nulla szerepel. Ez utóbbit onnan tudjuk meg, hogy alsó vagy felső háromszögmátrixról van-e szó. Jelen esetben a programunkban a főátló felett van kinullázva a mátrix.

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

int
main ()
{
    int nr = 5;
    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");
    }

    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;
}
```

Megoldás forrása:

A programunk elején itt is, a már jól megszokott módon include-oljuk a header fájlokat. Az stdio.h header-rel már korábban is találkoztunk, azt nem részletezném, viszont az stdlib.h sokak számára még új lehet. Ebben az esetben erre a `malloc` függvény miatt van szükség. Majd jön a deklaráció, ami most érdekesebb mint eddig valaha, hiszen deklarálunk egy double típusú pointerre mutató pointert. (8 bájtot foglal le)

Az if-es részben a `malloc` függvényt használjuk, ami helyet foglal a memóriában és egy pointert ad vissza. A `malloc` megkapja, hogy mekkora területet foglaljon le, ami esetünkben 5*8, azaz 40 bájtot kell lefoglalnia. (nr * double pointer mérete) Az if-el vizsgáljuk, hogy sikeresen megtörtént-e a helyfoglalás és, hogy a `malloc` visszaad-e a double*-okra egy mutatót. Ha ez nem volt sikeres, akkor kilép a programból. (Akkor következhet be leginkább, ha nincs elegendő tárhely) Amennyiben viszont sikeres volt, haladhatunk tovább a programon.

A következő szakaszban egy for ciklust látunk, ami 0-tól nr-ig megy, azaz 5-ig. Ezen belül memóriát

foglalunk a tm[i]-edik elemének (az elsőnek 8 bájtot, aztán 16-ot, 24-et és így tovább), amiről visszad a `malloc` egy pointert. Persze itt is ellenőrizve van, hogy sikeres-e a helyfoglalás és pointer létrehozás.

Elérkeztünk a mátrixunk megszerkesztéséhez. Két egymásba ágyazott for ciklus segítségével tudunk végigmenni az elemeken (mivel külön kell kezelni a sor- és oszlopindexeket) Itt feltöltjük a mátrixot, majd a következő, szintén egymásba ágyazott for ciklus segítségével iratjuk ki mátrix értékeit.

Majd láthatjuk, hogy a 4. sorban lévő értékeket, hogyan változtathatjuk meg. Ezután újra kiiratjuk a mátrixunkat és futtatás után kiderül, hogy tényleg megváltoztak az utolsó sorban az értékek. A program legvégén érünk el a feladat leírásában is említett `free` függvényhez. Ennek segítségével tudjuk felszabadítani már korábban lefoglalt memóriát.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Az XOR művelet bitenként hasonlít össze két operandust. Kétféle értéket adhat vissza, attól függően, hogy a vizsgált két bit megegyezik vagy sem.(Ha megegyeznek, akkor 0-t ad vissza, ha nem akkor 1-et) Jelen esetben ez a két operandus a forrás bemenet, amit titkosítani akarunk és egy kulcs, amire a titkosításhoz van szükség. (Ennek a feladatnak a másik része, amikor fel is törjük a kódot, szintén megtalálható a könyvben. - Hello, Caesar! C EXOR törő)

```
#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);
    }
```

```
}
```

Megoldás forrása:

Deklarálunk két char tömböt. Az elsőben tároljuk a kulcsot, a másikban a beolvasott karaktereket. Valamint definiálunk még két változót, a kulcs_indexet, amit növelve bejárjuk a kulcs tömböt, és az olvasot_bajtokban fogjuk tárolni a beolvasott bájtokat.

Az strlen segítségével tudjuk meg a kulcs hosszát, amit a kulcs_meret-ben tárolunk. Az strncpy-vel pedig a kulcs tömbbe másoljuk az argv[1]-et, ami a kulcs.

A while cikluson belül a `read` segítségével tudunk beolvasni a pufferbe, 256 bájtot (ennyi a BUFFER_MERET). Ez a beolvasott bájtok számát adja vissza. Ezen belül a for ciklusunk 0-tól megy a beolvasott_bajtok-ig és a buffer i-edik elemét össze EXOR-ozzuk a kulcs tömb megfelelő elemével. Végül kiiratjuk a buffer tartalmát.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása:

A fordításhoz telepítenünk kell egy csomagot: **sudo apt install default-jdk** A java kódot a következőképpen fordítjuk:

```
javac ExorTitkosító.java
```

Futtatás:

```
java ExorTitkosító 321cba > titkosított.szöveg
```

Itt 321cba lesz a kulcs és a "titkosított.szöveg" nevű fájlba irányítjuk a szöveget, amit titkosítunk. (Begépeljük, amit titkosítani szeretnénk, majd Ctrl+D)

A titkosító töréséhez szükség van a kulcsra. Az alábbi módon tudjuk törni a titkosított szöveget: (A standard output-ra írja ki az eredeti "tiszta" szöveget.)

```
java ExorTitkosító 321cba < titkosított.szöveg
```

A teljes program:

```
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();

        }

    }

}
```

Hasonló helyzetben vagyunk, mint az előző feladatban. Itt is titkosítanunk kell egy tiszta szöveget EXOR-ral, de most nem C-ben, hanem Java-ban.

Az egész kódunk egy jó nagy class-ból áll, aminek két nagyobb része van. Kezdjük talán a `main`-nel. A main itt máshogy néz ki, mint ahogy azt C-ben már megszoktuk. Ráadásul itt láthatunk először példát a kivételkezelésre. (try, catch) A `try` és a `catch` használata, nem csak Java-ban, hanem C++-ban is igen elterjedt Hiba esetén a try "dobja", a catch "elkapja" a hibát és küld egy hibaüzenetet a terminálba.

Az `ExorTitkosító` függvényen belül utasításokat hajtunk végre, olyanokat amiket már C-ben is csináltunk. Létrehozunk egy `byte`-okból álló tömböt. A `getBytes()` függvény segítségével olvassuk be a kulcsot a kulcs tömbbe. A buffer tömbnek ugyanúgy, ahogy a korábbi C-s feladatban, 256 bájtból álló területet foglalunk. Innen már nagyon hasonlóan működik a program, mint a C-s testvére. A `while` cikluson belül itt is található egy `for` ciklus, ahol elemenként össze EXOR-ozzuk a buffer tartalmát a kulccsal. Végül kiiratjuk a puffer tartalmát.

4.4. C EXOR törő

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

Megoldás forrása:

Ebben a feladatban pont az ellentétét kell csinálni, annak amit a második feladatban csináltunk. Az ott titkosított szöveget kell feltörnünk. Ez, érthető módon egy fokkal nehezebb feladat.

A teljes program:

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

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

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

  return (double) titkos_meret / sz;
}

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

  double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

```
  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;

  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;
}
```

A program elején lévő _GNU_SOURCE új lehet számunkra. Erre a strcasestr használata miatt van szükség. Láthatjuk, hogy a kulcsméret 8-ra van állítva, azaz feltételezzük, hogy a kulcs 8 elemből áll. (nem tűnik túl hatékonynak)

Az `atlagos_szohossz` függvénnyel kiszámítjuk a bemenet átlagos szóhosszát. Majd a `tiszta_lehet` függvény megvizsgálja, hogy a fejtésben lévő kód tiszta-e már. Itt elérkeztünk a programunk egy újabb gyengeségéhez, ugyanis a program feltételezi, hogy a tiszta szöveg valszeg tartalmazza a gyakori magyar szavakat, illetve az átlagos szóhossz vizsgálatával akarja csökkenteni a lehetséges töréseket. Ha ezeknek nem felel meg a tiszta szöveg, akkor nem tudjuk feltörni.

Az `exor` függvény hasonlóan működik, mint a titkosításnál. Ezáltal visszakapjuk a tiszta szöveget, elvégre ha valamit duplán EXOR-ozunk, akkor önmagát kapjuk. Majd elérünk az `exor_tores` függvényhez, ami 0-át valamint 1-et ad vissza, a szöveg tisztaságától függően. A `main`-en belül elvégezzük a szükséges deklarációkat, majd egy `while` ciklussal folyamatosan olvassuk a bájtokat, a bemenet végéig, vagy amíg a bufferünk tele nem lesz. A következő `for` ciklussal azokat a helyeket, amik megmaradtak a bufferben kinullázuk. Majd jön egy időigényes rész, ahol az összes lehetséges kulcsot előállítjuk. A végén meghívjuk az `exor_tores` függvényt, aminek ha 1 a visszatérési értéke, akkor kiírja a program a kulcsot és a már feltört szöveget. (megj.: csak azokat tudja feltörni, amiket számokkal kódoltunk)

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 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.