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 Tóth, Attila	2019. március 10.	

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!

Megoldás videó:

Megoldás forrása: https://github.com/Savitar97/Prog1/tree/master/vegtelen

Végtelen ciklust a legkönnyebben 3 féle képpen tudunk írni(ezek a standard formák):

```
#include <stdio.h>
int main(){
while(1);//végtelen ciklus while-al
for(;;);//végtelen ciklus for-al
do{}
while(1);//végtelen ciklus do while segítségével
return 0;
}
```

A végtelen ciklus következhet hibából, de van úgy, hogy szándékosan használunk végtelen ciklusokat például a programok menüjénél, de a program futása is egy végtelen ciklus, amelyet az X gombra kattintás szakít meg.Ha simán írunk egy végtelen ciklust az egy szálat fog kihasználni 100%-on, mindaddig amíg nem párhuzamosítjuk, ezt a:

```
#pragma omp parallel
```

segítségével érjük el.Ekkor a program már minden szálat képes kihasználni 100%-on. Fordítani pedig a következőképpen tudjuk:

gcc vegtelen3.c -o vegtelen3 -fopenmp

Ha azt akarjuk, hogy 0%-ot használjon a processzorból akkor azt a:

```
sleep();
```

használatával tudjuk elérni, amely lehetővé teszi, hogy a meghívott szálat egy meghatározott ideig "sleepeltesse". Az időt a () között adhatjuk meg másodpercben, ha 0-t adunk meg, akkor végtelen időre sleepeltethetünk. A sleep függvényt az:

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

könyvtár tartalmazza. Tehát a használatához megkell hívnunk.

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...

Ha a T100 as függvény létezne és megkapná a P-t paraméternek akkor igazat ad vissza. De ha a T100 asnak a T100 ast adjuk meg tehát rekurzívan hívjuk meg a T100 ast akkor azt írná ki, hogy a T100 asban nincs végtelen ciklus, pedig a bemenő argumentuma egy végtelen ciklus.

Tanulság nem lehet jelenleg olyan programot írni, amely normálisan eldönti egy másik programról, hogy kifog -e fagyni avagy sem.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:https://github.com/Savitar97/Prog1/blob/master/valtcser/valtcser.c

Két változó értékének felcserélése többféle módon is történhet a legalapvetőbb a segédváltozó használata.Ekkor a 2 változóhoz behozunk egy ideiglenes segédváltozót, amiben valamelyik változó értékét letároljuk, majd az első változó értékét egyenlővé tesszük a másodikkal, majd a második értékét egyenlővé tesszük az ideiglenesben letárolt első változó értékével. Ez itt látható:

```
#include <stdio.h>
int main(){
int elso=5,masodik=3,temp;
    temp=elso;
    elso=masodik;
    masodik=temp;
}
```

De ezen a módszeren kívül, lehetséges összeadás-kivonással,szorzás-osztással,vagy logikai kizáró vagy művelet segítségével felcserélni két változó értékét.

Legyen két változónk a és b. Összeadással az a-ba összeadjuk a-t és b-t. Majd az a-ból kivonjuk a b-t és ezt letároljuk b-be.Ekkor b értéke egyenlő lesz az a változó kezdeti értékével,majd az a-ból kivonjuk a b változót ekkor a értéke egyenlő lesz b kezdeti értékével, tehát felcserélődtek az értékek.Szorzás-osztással, ugyan így működik.

Két változó értékét logikai operátorral a kizáró vaggyal is megcserélhetjük.

```
#include <stdio.h>
int main(){
int elso=5,masodik=3;
    elso=elso^masodik;
    masodik=elso^masodik;
    elso=elso^masodik;
}
```

A kizáróvagy(xor) lényege, hogy csak akkor igaz ha az egyik igaz.Ez binárisan azt jelenti,hogy akkor 1 es ha ugyan azon a biten lévő érték az egyik változónál 1 es a másikban 0 ás.Az elso változó binárisan 0101 a masodik 0011 kizáró vagyot végre hajtva az elso értéke 0110 lesz, aminek az értéke 6. Majd újra kizáró vagyot végrehajtva a masodik értéke 0101 lesz, ami 5 tehát megkapta az elso ertékét. Ezután még egyszer kizáróvagy-ot használunk ekkor az elso értéke 0011 lesz, ami 10 es számrendszerbe 3. Tehát a két változó értéke felcserélődött.

2.4. Labdapattogás

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

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

Megoldás forrása:https://github.com/Savitar97/Prog1/tree/master/labda

Labdapattogtatás if-el: c-ben megadtam egy maximális méretet a pályának ez az x és y változó. A labda kezdetleges koordinátáit a labdax és labday-ban tárolom. Ezen kívül kell még 2 változó, amely a labda mozgásáért felelős ez a tempx és tempy. Magát a labdát karakterként a labda változóban tárolom.

```
        #include <stdio.h>
        int main(){
        char labda='o';
int x=80,y=15,labdax=1,labday=1,tempx=1,tempy=1;
        }
```

A labdapattogtatást a for(;;) végtelen ciklus és egy rajzol eljárás folytonos meghívása szolgálja. A labda mozgását a koordináták temp-el való növelése szolgálja. Az if-ek segítségével érem el, hogy ha a labda eléri a pálya szélét, akkor a temp előjele változzon, így az érték csökkeni kezd, majd csökkenés után ha újra eléri a pálya szélét a -1-szeres szorzással újra pozitívba vált. A késleltetett kiírást az usleep éri el, az értéket microsec-be kell megadni és az unistd.h könyvtár tartalmazza ezt a függvényt.

```
        #include <stdio.h>
        #include <unistd.h>
        int main(){
        for(;;)
{
  labdax+=tempx;
  labday+=tempy;
  if(x-1<=labdax)
  {
    tempx*=-1;
  }
  else if(y-1<=labday)
  {
    tempy*=-1;
  }
  else if(labdax<0)
  {
    tempx*=-1;
  }
  else if(labday<0)
  {
    tempy*=-1;
  }
  rajzol(labdax,labday,labda);
  usleep(100000);
}
        }
```

If nélkül azt, hogy a labda vissza pattanjon a maradékos osztás végzi el és az abszolút érték.

```
                    x=abs(szelesseg-lepteto%(2*szelesseg));
                    y=abs(tmagassag-lepteto%(2*tmagassag));
                    lepteto++;
```

És persze szükséges mellette egy változó aminek az értékét folyamatosan növeljük és a pálya méretének 2x esével osztjuk el maradékosan, majd kivonjuk a pálya méretéből és ez határozza meg a labda koordi-

nátáját.Tehát mondjuk egy 50 es pálya méretnél 50-1%100=49 ... így csökken egészen 0-ig majd mikor a léptető eléri az 51 et 50-51=-1 el, de ennek az abszolút értéke 1, tehát újra növekedni fog.

Tapasztalat:C-ben van egy kis hiba mivel, amikor eléri a pálya tetejét a labda nem egyből pattan vissza, ez csak if-nél jelentkezik. If nélkül a két abszolút értékes függvénnyel nem jelentkezik ez a hiba.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása: https://github.com/Savitar97/Prog1/blob/master/szohossz/bitshift.cpp

Tanulságok, tapasztalatok, magyarázat...

A szóhossz megnézéséhez a bitenkénti léptetés operátort használjuk:

```
  while (szam <<=1){
cout<<szam<<'\n';
counter++;
  }
```

Ez annyit jelent hogy az egyest egyre jobban balra toljuk és jobbról 0-ákkal pótoljuk.

Ez azt eredményezi, hogy 2-nek hatványait kapjuk és amikor eléri az int maximális méretét utána 0-t kap eredményül, mivel a bitsorozat teljesen ki 0-ázódik.Tehát a 32. lépésre, nem lesz olyan bit, amin képesek leszünk ábrázolni az 1 est.

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
2147483648
Az egy 31 lepes utan lesz 0.
```

Igazából 32 bites a szóhossz csak az elsőt nem számolja szóval 31+1.

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: https://github.com/Savitar97/Prog1/blob/master/pagerank/pagerank.c

Tanulságok, tapasztalatok, magyarázat...

A pagerank algoritmust a google találta ki azért, hogy megkönnyítsék a weben való keresést. Maga a pagerank egy számba sűríti a weblap értékét. A lényege, hogy minél több oldal mutat a weblapunkra annál értékesebb. Ez azért van mert, úgy gondolták a google alapítói, hogy a weboldal készítői, azért linkelnek be oldalakat mert hasznosnak találják.

Ez felfogható úgy is, mint egy választás. És minden oldal képes szavazni és az, hogy valaki a mi linkünket használja olyan mintha ránk voksolna és akinek több a szavazata az van előrébb a rangsorban.

Első lépésként megadjuk a kapcsolati gráfot. Tehát, hogy melyik oldal melyik oldalra mutat. Ezt egy mátrixban tároljuk le, mivel 4 honlappal dolgozunk, ezért egy 4x4 es mátrix lesz:

```
        #include <stdio.h>
        #include <unistd.h>
        int main(){
        double graf[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}
                            };
}
        }
```

Azért double mivel a pagerank nem feltétlenül csak egész szám lehet.

Ezután létrehozunk még 2 db egydimenziós tömböt. Az egyikben a végleges pagerankot tároljuk míg a másikban az ideiglenest. Az ideiglenes vektorban minden oldal pagerankját 1/4-re állítjuk mivel 4 oldal van.

Majd indítunk egy végleges ciklust,amely addig fut, amíg a pagerank kisebb nem lesz, mint a dumping factort az az a csillapító értéket, most ez 0.00001-ben lett meghatározva

A végtelen ciklus elején áttöltjük az ideiglenes pagerankból az értékeket a végleges pagerank tömbünkbe.

Ezt követően indítunk egy egybeágyazott for ciklust, amely a letárolt kapcsolati gráfos tömb(a szétosztott szavazatokat tárolja) sorait megszorozza a jelenlegi pagerankkal és a sorok összegét, mármint egyessével egy értékbe tömöríti és azt betölti az ideiglenes pagerankba és ez addig folytatódik, amíg kinem lép a végtelen ciklusból.

```
        #include <stdio.h>
        #include <unistd.h>
        #define dumping_factor 0.0001

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

```
            {
                for(i=0;i<4;i++)
    {
      PR[i] = PRt[i];
    }
                for (i=0;i<4;i++)
    {
      double temp=0;
      for (j=0;j<4;j++)
        temp+=graf[i][j]*PR[j];
      PRt[i]=temp;
    }

                if ( dif(PR,PRt, 4) < dumping_factor)
    break;
            }
  }
        }
```

A távolság függvény paraméterként megkapja a végleges pagerankot és az ideiglenest és, hogy hány db oldal van. Majd kivonja a pagerank i-edik eleméből az ideiglenes pagerank i-edik elemének értékét és ezek abszolút értékét összeadja a tav nevű változóban, amely a függvény visszatérési értéke lesz.

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

        double dif(double pagerank[],double pagerank_temp[],int db)
{
  double  dif= 0.0;
  int i;
  for(i=0;i<db;i++)
    {
      dif +=fabs(pagerank[i] - pagerank_temp[i]);
    }
  return dif;
}
```

Érdekesség ha az egyik oldal nem mutat semmire.Tehát ha az utolsó oszlopot mondjuk teljesen ki 0-ázzuk akkor a pagerank is kinullázódna ha 2 tizedes jegyik néznénk az értéket.

```
PageRank [0]: 0.000010
PageRank [1]: 0.000047
PageRank [2]: 0.000026
PageRank [3]: 0.000010
```

A helyes megoldás:

```
PageRank [0]: 0.090908
PageRank [1]: 0.545453
PageRank [2]: 0.272730
PageRank [3]: 0.090908
```

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

Prímnek nevezzük azokat a számokat, amelyek csak 1-el és önmagukkal oszthatók.Ikerprímek azok a prímszámok, amelyek különbsége 2.

A program egy megadott x értékig kikeresi a prímeket. Majd megnézi a köztük lévő diferenciát (diff),ahol ez a diferencia 2, annak az indexét egy tömbben(idx) tárolja(de csak az ikerprímpár első tagjának indexét, ezért kell a t2primes-nál a +2) tehát a prímek közül kiszűri, hogy melyek ikerprímek.

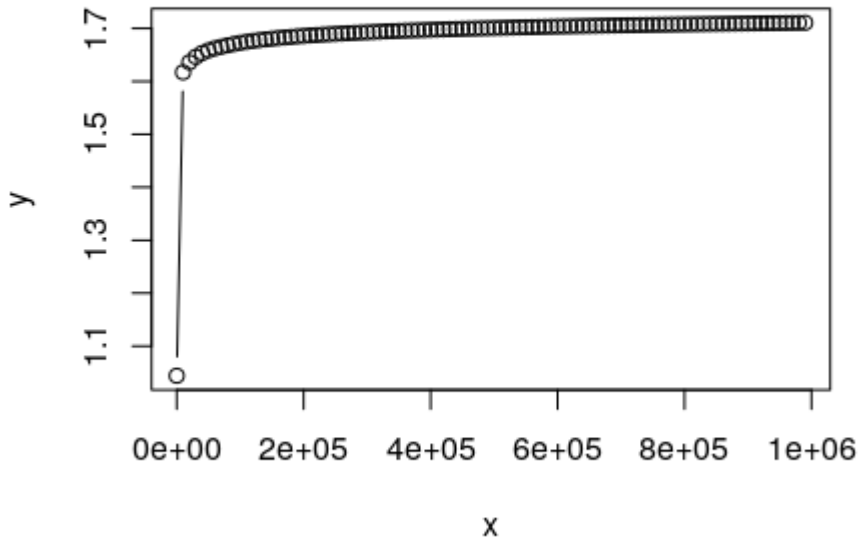
```
 primes = primes(x)
diff = primes[2:length(primes)]-primes[1:length(primes)-1]
idx = which(diff==2)
t1primes = primes[idx]
t2primes = primes[idx]+2
```

Majd az rt1plust2-ben összeadjuk ezeknek a reciprokát végül a függvényünk visszatérési értéke az rt1plust2-nek az összege.

A seq függvény hasonló a for ciklushoz seq(from, to, by=), from,hogy mettől(13) to, hogy meddig(1000000) és a by, hogy milyen lépésszámmal(10000). Ez határozza meg az x tengely beosztását.

A sapply függvény az x ekhez rendeli egyessével az stp függvényben kapott értékeket y-ként.

Végül a plot kirajzolja a függvényt.



A képen látható, hogy a párosprímek reciprokának összege egyre jobban tart a 2 felé,tehát egy véges értékhez konvergál, amely a Brun konstans azaz a Brun tétel teljesül.

De ezzel még mindig nem tudjuk eldönteni, hogy végtelen vagy véges számú prímszám van mert úgysem lépik át ezt a határt csak megközelítik.

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...

A Monty Hall probléma lényegében 3 ajtó közül kell kiválasztanunk a nyerteset viszont ha nem találjuk el akkor újra kezdhetjük.

Annak a valószínűsége, hogy egyből a jó ajtót találjuk el 1/3 és az, hogy rosszat 2/3. Viszont a választás után a 3 ajtó közül a műsorvezető kinyit egyet, amelyik mögött nem a nyeremény van.(A játékvezető tudja melyik ajtó mögött van a nyeremény) Majd ezután megkérdi a játékost, hogy szeretne -e változtatni a választásán. Elvileg az ajtó nyitása után a nyerési arányunk redukálódik 1/2-re, hogy nyerünk és 1/2-re, hogy vesztünk. A nagy kérdés itt az, hogy megéri -e váltanunk.Ez a program pont ezt szimulálja.

A kísérletek száma változóban definiáljuk, hogy hányszor fusson le a kísérlet.Azaz a minták száma.

A kiserlet és a jatekos tömbök, amelyeket 1 és 3 közé eső számokkal tölt fel a sample. A műsorvezető egy vektor amelyet ugyan olyan méretűre deklarálunk mint a kísérletek száma.

Egy for ciklussal bejárjuk a tömböt és ha a játékos eltalálja, hogy melyik ajtó mögött van akkor a mibol tömbbe a másik két ajtó lehetősége kerül.Ha nem találja el akkor csak egyetlen érték az az ajtó ami mögött nincs semmi,de nem választotta a játékos.

Ezután a műsorvezető úgymond kinyit egy ajtót tehát választ egyet a mibol tömbből.

Majd lefut egy feltétel, amely megmutatja hányszor nyerne a játékos ha nem változtat.Tehát a tömbbe azok az indexek kerülnek amikor a jatekos és a kiserlet megegyezik.

Létrehozunk egy új vektort amiben megváltoztatjuk a választást úgy, hogy kivételként adjuk a műsorvezető által kinyitott ajtót és a játékost által választottat.

Végül lefuttatunk egy ugyan ilyen feltételes vizsgálatot, hogy mi lett volna ha mindig változtatunk. És itt is ugyan úgy letároljuk egy tömbbe, hogy mely indexeknél nyert a játékos. És kiiratjuk a statisztikát, amely megmutatja, hogyan járnánk jobban ha mindig változtatnánk vagy ha tartózkodnánk az először választott ajtóhoz.

Egy példa a program futására:

```
> valtoztatesnyer = which(kiserlet==valtoztat)
>
>
> sprintf("Kiserletek szama: %i", kiserletek_szama)
[1] "Kiserletek szama: 100"
> length(nemvaltoztatesnyer)
[1] 34
> length(valtoztatesnyer)
[1] 66
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.5151515
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 100
>
```

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Decimálisból unárisba, úgy váltunk, hogy annyi 1 est írunk le,amennyi a szám értéke vagy másképp fogalmazva amilyen messze van a 0-tól.Pl.:n=90 esetén 90 db 1 est kell leírnunk.

Tehát itt pozitív számokat tudunk ábrázolni. A megvalósítás 2 féle lehet vagy indítunk egy for ciklust 0-tól és minden egyes lépésnél egy stringbe összefűzzük az egyeseket. Vagy pedig a számtól indítunk egy ciklust 0-ig és ugyan ezt tesszük.

3.2. Az $a^nb^nc^n$ 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: https://github.com/Savitar97/Prog1/blob/master/hivatkozas/hivatkozas.c

Tanulságok, tapasztalatok, magyarázat...

A C utasítások a C nyelv kulcsszavai. A C nyelv tartalmazza a többsoros utasítás blokkokat,iterációkat (for,while,do-while),vezérlő szerkezeteket(if,switch),operátorok(++,--,!=,stb),deklarációk.

Backus normal form egy általánosított leírása a programozási nyelveknek.Nyelv független.

```
    A backus leírás röviden:
<szimbólum> ::= <kifejezés a szimbólumra>
    van egy szimbólum aztán a ::= után van egy formai leírása
<egész szám> ::= <előjel><szam>
<előjel> ::= [-|+]
<szam> ::= <szamjegy>{<szamjegy>}
<szamjegy> ::= 0|1|2|3|4|5|6|7|8|9
```

A c89-ben még nem lehet egysoros kommenteket írni (//) és szintén nem lehet a for ciklus fejében változót deklarálni, amit a c99 már enged. A különböző változatoknál a fordítást a **-std** kapcsolóval érjük el ez, így néz ki a gyakorlatban:

gcc -o hivatkozas -std=c89 hivatkozas.c

gcc -o hivatkozas -std=c99 hivatkozas.c

```c
#include <stdio.h>
int main(){
int i;
for(i=0;i<10;i++)
/*Ez így lefog futni c89-ben is.
  Viszont
  for(int i=0;i<10;i++)
  ez nem.
*/
return 0;
}
```

A következő hibaüzenetet kapjuk:

```
hivatkozas.c: In function ‘main’:
hivatkozas.c:5:2: error: C++ style comments are not allowed in ISO C90
  //ahahahah
  ^
hivatkozas.c:5:2: error: (this will be reported only once per input file)
hivatkozas.c:6:2: error: ‘for’ loop initial declarations are only allowed in C99
 or C11 mode
  for(int i=0;i<10;i++)
  ^~~
hivatkozas.c:6:2: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11
to compile your code
```

Emellett van olyan ami a c89-ben működik, de c99 ben nem. Ilyen a következő kódcsipet:

```c
#include <stdio.h>


int main()
```

```
{

int restrict=1;
if (restrict) printf("restricted");
return 0;
}
```

Azért fordulhat le mivel a restrict még nem kulcsszó c89-ben,viszont c99-ben már igen. A restrict megadja, hogy mely pointerek férhetnek hozzá az adott memória területhez,ezeket a pointereket nem lehet módosítani.

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: https://youtu.be/9KnMqrkj_kU

Megoldás forrása: https://github.com/Savitar97/Prog1/tree/master/real

Tanulságok, tapasztalatok, magyarázat...

A lexernél az egyes részeket %%-jelek választják el. Az első résznél jön a könyvtár hivatkozás és a deklarációk és az első rész végén definiálunk(ilyen itt a digit amiben a számokat definiáljuk) a definícióknál lehet megadni a karakter csoportokat, amelyeket keresni akarunk a beolvasott szövegből.

A következőben jöhetnek a formázási szabályok itt mondhatjuk meg, hogy mi történjen ha megtalálja az adott karakter sorozatot vagy karaktert a lexer.Itt már használhatjuk a definíciókat.

Az első kapcsos zárójelben megadtuk, hogy számot keresünk, ezután a csillag azt jelenti,hogy bármennyiszer előfordulhat 0 vagy akárhány.Majd egy pontnak kell követni azután a + miatt jönnie kell egy számnak legalább vagy többnek. A kérdőjel viszont azt jelzi, hogy nem muszáj pontnak következnie és utána számnak ez azért kell mivel az egész számok is beletartoznak a valós számokhoz.

Majd azt adjuk meg ha találunk ilyen karaktersorozatot akkor a countert növeljük-1 el. És írjuk ki ezt a karakter sorozatot -között az atof függvény pedig ezt a karaktersorozatot valós számmá konvertálja.

A programot a következő képpen kell fordítanunk:

lex -o real.c real.l

Ilyenkor a lexer megírja a c programot, majd a létrehozott .c fájlt gcc-vel fordítjuk.

gcc -o real real.c -lfl

Az utolsó részben jön a program main része itt meghívjuk a yylex() függvényt és kiirassuk,hogy hány db valós számot találtunk.

A programot a **Ctrl+D**-vel tudjuk leállítani.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.6. A források olvasása

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

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

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

Bugok

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

---

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

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

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

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

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

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

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

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

1. Bármely x-hez létezik olyan y, hogy az y nagyobb mint x és y prím. 2. Bármely x-hez létezik olyan y, hogy y nagyobb mint x és y prím és y+2 is prím. 3. Létezik olyan y, hogy bármely x esetén ha x prím akkor az x kisebb mint y. 4. létezik olyan y, hogy bármely x esetén ha y kisebb mint x akkor x nem prím.

Az első állítás azt mondja ki, hogy a prímszámok száma végtelen. Míg a második azt jelenti, hogy végtelen sok ikerprím létezik.Itt az SSy a successor function vagy másnéven a rákövetkező függvény,tehát **S(S(y))=(y+1)+1)**.

A 3. állítás ennek az ellenkezőjét fejezi ki, tehát azt, hogy a prímszámok száma véges. A negyedik állítás ezzel ekvivalens, mivel azt mondja ki, hogy létik olyan y amitől nincs nagyobb prímszám.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

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

Mit vezetnek be a programba a következő nevek?

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

Megoldás videó:

Megoldás forrása: https://github.com/Savitar97/Prog1/tree/master/deklaracio

Tanulságok, tapasztalatok, magyarázat...

Az int a létrehoz egy változót, amely int típusú az az egész, a néven. Egy változónak van neve,típusa,hatásköre,mer
címe ahol tárolódik,értéke.

int *b létrehoz egy mutatót, amely a-nak a memória címére hivatkozik.

Az r rekurzívan hivatkozik a értékére. Vagyis ugyan arra a memória területre hivatkozik mint az a. Tehát ha r értékét változtatjuk akkor a-nak az értéke is változik. Szemléltetésként a következő kód szolgál:

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

using namespace std;

int main()
{
  int a=5;
```

```
  int &r=a;
  int *d=&a;
  int *b=&r;
  cout<<a<<'\n'<<r<<'\n'<<d<<'\n'<<b<<'\n';
  r=r+2;
  cout<<a<<'\n'<<r<<'\n'<<d<<'\n'<<b<<'\n';
  return 0;
}
```

Futtatva következő eredményt kapjuk:

```
5
5
0x7fffc06a097c
0x7fffc06a097c
7
7
0x7fffc06a097c
0x7fffc06a097c
```

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

A c deklarál egy 5 elemű tömböt.

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

A tr tömb rekurzívan hivatkozik a c tömbre.

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

Létrehoz egy 5 elemű mutató tömböt.

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

Olyan függvény ami egy egészre mutatót ad vissza.

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

Egy egészre mutató mutatót ad vissza a függvényt.

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 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.