Ez a wikioldal 2007 februárjától a BME alkalmazott matematikus hallgatói számára oktatott Informatika 2 (más néven SZIMP2) tárgy honlapja. A tárgy oktatásához felhasznált, http://wiki.math.bme.hu/ -n belüli wikioldalak GNU FDL licenc vagy (választás szerint) CC-BY-SA-2.0 licenc szerint szabadon használhatók és terjeszthet?k.

# Tartalomjegyzék


- 1 Oktatók
- 2 Hallgatók
- 3 Tárgykövetelmények
- 4 Katalógus
- 5 A hallgatók munkája
- 6 Ajánlott szoftverek
- 7 Tananyag
- 8 BarbaLinux
- 9 Konvenciók
- 10 A gcc-parancssor elemeinek jelentése
- 11 1. gyakorlat (2007-02-13 és 2007-02-16)
- 12 1. el?adás (2007-02-16)
- 13 2. gyakorlat (2007-02-20)
- 14 2. el?adás (2007-02-23)
- 15 3. gyakorlat (2007-02-27, 2007-03-02)
- 16 3. el?adás (2007-03-02)
- 17 4. gyakorlat (2007-03-06, 2007-03-09)
- 18 4. el?adás (2007-03-09)
- 19 5. gyakorlat (2007-03-10, 2007-03-13)
- 20 5. el?adás (2007-03-10)
- 21 6. gyakorlat (2007-03-20, 2007-03-23)
- 22 6. el?adás (2007-03-23)
- 23 7. gyakorlat (2007-03-27, 2007-03-30)
- 24 7. el?adás (2007-03-30)
- 25 8. gyakorlat (2007-04-03, 2007-04-06)
- 26 Ami a C-s el?adásokból kimaradt
- 27 8. el?adás és további el?adások
- 28 9. gyakorlat és további gyakorlatok
- 29 Az els? ZH
    - 29.1 Id?pontok (pótZH is)
    - 29.2 Eredmények
    - 29.3 Az elméleti rész
    - 29.4 A programozási feladat
- 30 Házi feladatok
    - 30.1 Technikai részletek
    - 30.2 Kiírt házi feladatok

- 30.3 Típushibák a házikban


# Oktatók

- Az el?adásokat és a gyakorlatokat tartja Szabó Péter <pts+i@math.bme.hu>, beceneve pts.
- A házi feladatokat és a ZH-kat javítja, a konzultációk egy részét tartja Sisak Áron <aron@math.bme.hu>.

Egyéb személyek:

- Tárgyfelel?s Wettl Ferenc <wettl@math.bme.hu>.
- A Live CD-t készíti Er? Zsolt <zsero@math.bme.hu>.
- Az el?z? félévben Pröhle Péter és Rácz Balázs oktatta a SZIMP2-t, akkor ez volt a tárgy honlapja: http://www.ilab.sztaki.hu/~bracz/p/cimp2_2006tavasz.html

# Hallgatók

A gyakorlati ismeretek kell? elsajátításához elengedhetetlen, hogy a gyakorlati kurzuson mindenki saját számítógéppel dolgozzon. Az H.27-es laborban 14 munkára alkalmas gép van, amib?l hallgatók számára kényelmesen használható 12 db. A biztonság kedvéért az egyik gépet hagyjuk meg tartaléknak, tehát van 11 gép, 3 gyakorlati kurzus, és összesen 33 hallgató, akiket a fentiek miatt 3 egyenl? részre kell osztani. A felosztás 2007-02-16 (pénteken) 9:10-kor, az el?adás el?tti 5 percben fog megtörténni. Két lehetséges megvalósuás van.

1. A hallgatók egyeztetnek, majd összeállítanak egy nekik megfelel? felosztást, és azt írásban benyújtják a jelzett id?pontban. A formai követelmények: 3 lista lesz, melyeken a tárgyat felvett hallgatók felhasználónevei vannak egyértelm?en szétosztva. Minden hallgató pontosan egy listán szerepel. Minden listán minimum 11, maximum 12 felhasználónév található.
2. Ha a fenti felosztás nem kerül benyújtásra a jelzett id?pontban, akkor az *el?zetes gyakorlati kurzusbeosztás* lép életbe (lásd lent).

Indokolt esetben utólag is lehet cserélni (2 hallgató egymás között cserél). A csere módja: a hallgatók egymással egyeztetnek, majd mindketten küldenek pts-nek és a másik hallgatónak egy e-mailt legalább 24 órával a korábbi gyakorlat el?tt.

A Neptunban lev? hallgatói kurzus-hozzárendelés módosításának intézésével a hallgatóknak nem kell tör?dniük.

Aktuális gyakorlati kurzusbeosztás (közös megegyezéssel):

- végleges kedd 14:15-re:
  - bartazsu Barta Zsuzsanna
  - btimi Borsos Katalin
  - czirakit Cziráki Tamás
  - zsero Er? Zsolt
  - farbas Fárbás Tamás
  - kristofh Hörömpöly Kristóf
  - jgabor Janecskó Gábor
  - macsakos Mácsai Ákos
  - molnarg Molnár Gábor
  - semdan Semler Dániel
  - tbarbara T?ke Barbara
  - wlaci Winkler László

- végleges kedd 15:15-re:
    - ♦ szape Szabó Péter
    - ♦ szbence Sz?cs Benedek
    - ♦ agiesze Esze Ágnes
    - ♦ daraib Darai Borbála
    - ♦ diagy Gy?ri Klaudia
    - ♦ dsziraki Sziráki Dorottya
    - ♦ hutivi Hutvágner Ivett
    - ♦ jocika Pinczés József
- végleges péntek 8:15-re: (a *kivéve* munkanapok helyett ugyanazon héten kedd 15:15-re kell jönni)
    - ♦ csizbal Csizmadia Balázs (kivéve február 23., március 2.)
    - ♦ gszj Göbölös-Szabó Julianna (kivéve március 2., március 9.)
    - ♦ vhalasz Halász Veronika (kivéve március 9., és március 23.)
    - ♦ istvanko Kolossváry István (kivéve március 23., és március 30.)
    - ♦ rkozma Kozma Róbert Thijs (kivéve március 30., április 6.)
    - ♦ nagyal Nagy Attila (kivéve április 6., április 13.)
    - ♦ pintye Pintye Norbert (kivéve április 13., április 20.)
    - ♦ sepsir Sepsi Róbert (kivéve április 20., április 27.)
    - ♦ tuzcsaba T?z Csaba (kivéve április 27., május 4.)
    - ♦ urbane Urbán Eszter (kivéve május 4., május 11.)
    - ♦ szabov Szabó Viktor (kivéve május 11., május 18.)
    - ♦ fukoildi F?k? Ildikó (kivéve május 18., feburár 23.)
    - ♦ csata Csata Árpád (kivéve március 2., március 9.)
    - ♦ gubeka Gubek Andrea (kivéve március 9., március 23.)
    - ♦ kbotond Koszta Botond (kivéve március 23., és március 30.)
    - ♦ mikulan Mikulán Attila (kivéve március 30., április 6.)
    - ♦ ivajda Vajda István

# Tárgykövetelmények

A szorgalmi id?szakban teljesítend?:

- A kiadott házi feladatok megoldása és beadása a SIO rendszerbe. Csak azt a feladatot kell beadni, amelynek a beadási határideje ezen a wikioldalon szerepel. Addig kell próbálkozni, amíg a rendszer OK-val el nem fogadja a megoldást. Az el nem fogadott megoldás és a meg sem kisérelt megoldás is 0 pontot ér feladatonként. Az elfogadott megoldás 1 pontot ér.
- Az els? ZH megírása. A ZH valószín?leg papíralapú lesz. Az els? ZH témája: C programozási nyelv és egyszer? algoritmusok. Csak azt kell tudni, ami el?adáson vagy gyakorlaton elhangzott. Puskát lehet majd használni. Lásd még a #Az els? ZH c. szakaszt.
- A második ZH megírása. A ZH valószín?leg számítógéppel megoldandó lesz. A második ZH témája: Ruby programozási nyelv és objektum-orientált programozás. Csak azt kell tudni, ami el?adáson vagy gyakorlaton elhangzott. Puskát lehet majd használni.

A félévközi jegy a fentieket összesítve kerül meghatározásra. Elégtelen osztályzatot kap,

- aki a gyakorlatok több, mint 30%-áról hiányzott, és a 30% fölötti hiányzásokat nem tudja igazolni;
- aki a jegybeiratásig (de legkés?bb a szorgalmi id?szak végéig) nem adta be OK-val elfogadva az összes kötelez? házi feladatot;
- aki az els? ZH-t vagy a hozzá tartozó pótZH-t nem teljesítette legalább elégségesre;
- aki a második ZH-t vagy a hozzá tartozó pótZH-t nem teljesítette legalább elégségesre.

Aki nem elégtelen osztályzatot kap, annak az osztályzata a ZH-kra kapott pontszámok lineáris kombinációjából számítódik.

Ha a jegybeiratás el?tt kiderül, hogy valaki nem maga készítette a házi feladatát, vagy elkészítette valaki más házi feladatát, akkor az érintett házi feladat egyik félnek sem fogadható el (hiába OK a SIO-ban), és fegyelmi eljárás indul az érintettek ellen a TVSz szerint.

Az, hogy valaki nem maga készítette a házi feladatát, kiderülhet például így:

- Gyakorlaton a gyakorlatvezet? megmutatja az OK-val elfogadott házi feladat forráskódját, és rákérdez egy részletre. Ha az illet? nem tud elfogadható választ adni, akkor er?s a gyanú, hogy nem maga készítette a házi faladatát.
- Ha az illet? ZH-n nem tud egy olyan feladatot megoldani, ami nagyon hasonló az korábban OK-val elfogadott házi feladatához (esetleg annál jóval egyszer?bb), akkor er?s a gyanú, hogy nem maga készítette a házi feladatát.
- Ha két, különböz? hallgató által beadott házi feladat-megoldás forráskódja nagyon hasonló, akkor er?s a gyanú, hogy az egyik fél nem maga készítette a házi feladatát.

# Katalógus

A gyakorlatok látogatása kötelez?, minden gyakorlaton van katalógus.

Az el?adások látogatása nem kötelez?.

# A hallgatók munkája

El?adáson a hallgatók figyelnek és jegyzetelnek. Legalább annyit leírnak, amennyi a táblára kerül.

Gyakorlaton a gyakorlat ideje alatt a hallgatók megcsinálják a kit?zött feladatot. Minden hallgatónak saját számítógépe van, amivel a feladatot megoldja. A gyakorlatvezet? segít, és ellen?rzi is a megoldást. Amelyik hallgató kész van, és unatkozik, plusz feladatot kér vagy talál ki magának, és azt is megoldja. Gyakorlaton általában nem kell jegyzetelni. Füzetbe jegyzetelés helyett ajánlott tömören, de egyértelm?en fájlba írni. A begépelend? kódrészleteket a hallgatók megkapják fájlban.

Gyakorlaton a hallgatók úgy ülnek le, hogy minden kezd? mellé üljön egy profi. Profinak számít az, aki az alábbiakból sokat tud: Linux grafikus felületek haladó felhasználói szint? ismerete, Linux parancssor felhasználói szint? ismerete, hatékony munka Linux parancssorban, hatékony munka Midnight Commanderben, angol nyelvtudás, jó hibadiagnosztizálási és hibajavítási készség, az órán kívül is gyakran használ Linuxot, tetsz?leges programozási nyelv hobbi-szint? ismerete, tetsz?leges programozási nyelv munkaer?piacon versenyképes ismerete, ismeretlen program m?ködésének megértése weben fellelhet? információk alapján, ismeretlen program m?ködésének megértése man page alapján, tetsz?leges program telepítése Windowsra, tetsz?leges program telepítése valamely Linux-disztribúcióra. A profinak nem kell ismernie se a C nyelvet, se a Ruby nyelvet.

A hallgatók otthon tanulnak, például az ajánlott irodalomból, esetleg az órai jegyzetükb?l. A ZH-kra való felkészülés legjobb eszköze a gyakorlás: az elmélet bebiflázása helyett sokkal eredményesebb (a ZH és a jöv? szempontjából is) egy adott feladatot adott módon megoldó program elkészítése, vagy ilyen példaprogramok tanulmányozása.

A hallgatók egyénileg (pl. otthon vagy a számítógép-laborban) megcsinálják a házi feladatokat, és beküldik a SIO rendszerbe. A beküldéshez internetelérés (web) szükséges.

A hallgatók megírják a ZH-kat.

A hallgatók a félév végén beiratják a jegyet.

# Ajánlott szoftverek

C nyelvhez Linux alatt ajánlott:

- sima szövegszerkeszt?: kate
- fordítóprogram: gcc
- fejleszt?környezet: Kdevelop
- egyéb segédprogramok: strace, valgrind

C nyelvhez Windows alatt ajánlott:

- fejleszt?környezet: Dev-C++: http://www.bloodshed.net/dev/devcpp.html Ezt kell letölteni: Dev-C++ 5.0 (9.0 MB) with Mingw/GCC.

Ruby nyelvhez Linux alatt ajánlott:

- sima szövegszerkeszt?: kate
- interpreter: ruby
- fejleszt?környezet: FreeRIDE http://rubyforge.org/frs/?group_id=31 freeride-linux-installer-0.9.6.sh

Ruby nyelvhez Windows alatt ajánlott:

- fejleszt?környezet: FreeRIDE http://rubyforge.org/frs/?group_id=31 freeride-win-installer-0.9.6-1.exe

Er? Zsolt egy Live CD-t készít, amin a fenti linuxos szoftverek megtalálhatók, és hosszadalmas telepítés nélkül kipróbálhatók. Barbalinux

# Tananyag

- algoritmusok C nyelven
- objektum-orientált programozás Ruby nyelven

Ajánlott irodalom:

- Standard C reference (2 oldalas) http://www.math.bme.hu/~pts/szimp2/stdc2.pdf
- Brian W. Kernighan--Dennis M. Ritchie: A C programozási nyelv. Az ANSI szerint szabványosított változkat. Kapható a jegyzetboltban.
- ruby.hu http://www.math.bme.hu/~pts/ruby.hu/1_eloszo.html (korábban innen volt elérhet?: http://segabor.web.elte.hu/ruby.hu/1_eloszo.html )
- Windowson interaktív, angol nyelv? Ruby tutorial: http://hacketyhack.net/

(Letölt, telepít, elindít.)

Egyéb irodalom:

- Ruby alaposztályok összes metódusa, angol nyelv? referencia: http://www.ruby-doc.org/core/
- egyéb leírás Rubyról: Ruby User's Guide (régi): http://www.math.hokudai.ac.jp/~gotoken/ruby/
- rövid összefoglaló Rubyról: Ruby Quick Reference (Ruby QuickRef): http://www.zenspider.com/Languages/Ruby/QuickRef.html

# BarbaLinux

Azok számára, akik a C programozást a gyakorlatokon megszokott környezetben (Kate és GCC) szeretnék gyakorolni, Er? Zsolt készített egy CD-t, amely tartalmazza ezeket a programokat. Köszönet érte!

Részleteket a Barbalinux wiki oldalán találhattok itt: Barbalinux

# Konvenciók

A konvenciók mind a hallgatók, mind az oktatók számára kötelez?en betartandók.

- A tárggyal kapcsolatos fájlokat mindenki a ~/szimp2 mappába helyezze.
- Minden fájlnév és mappanév kisbet?s.
- A fájlnevek ékezetes bet?ket nem tartalmaznak.
- A forrásfájlok (*.c és *.rb) ékezetes bet?ket nem tartalmaznak.
- Minden program forráskódja külön (*.c vagy *.rb) fájlba kerül. Tehát nem írunk felül egyetlen korábban létrehozott fájlt sem, hanem lemásoljuk a fájlt, és más néven mentjük el. A más néven mentés még a fájl tartalmának módosítása el?tt történik.
- A sor elején lev? szóközök számának megállapításánál betartjuk a *beljebb kezdési szabályt* (C nyelvhez lásd az 1. gyakorlat anyagában).

# A gcc-parancssor elemeinek jelentése

Ezt nem kell tudni, csak egy hallgató kérésére, érdekességképp került fel a wikibe. Az alábbi parancsot elemezzük:

```
$ gcc -W -Wall -s -O2 -lm -o <program> <program>.c
```

- *gcc*: a fordítóprogram neve. A *GNU C Compiler* és egyben a *GNU Compiler Collection* rövidítése. Házi feladat a Wikipédián utánanézni, mi az a GNU.
- *-Wall*: a legfontosabb figyelmeztet? üzeneteket (warning) bekapcsolja
- *-W*: még néhány fontos figyelmeztet? üzenetet bekapcsol
- *-s*: a fölösleges részeket (pl. nyomkövetési információk és szimbólumok) eltávolítja kimenetb?l. Nélküle nagyobb lenne a kimenet.
- *-O2*: bekapcsolja a második szint? optimalizálást. A harmadik szint? azért rossz, mert a kód lassan fordul, és túl nagy lesz. Az els? szint? azért rossz, mert a kód túl lassú lesz, és esetleg túl nagy is. Vannak egyéb szintek is, például alapból a nulladik szint érvényes. Nem nulla, hanem nagy O bet? van a kapcsolóban.
- *-lm*: a matematikai függvényeket (pl. *sqrt* és *cos*) tartalmazó matematikai függvénykönyvtárat teszi elérhet?ve. Emellett a forráskódban szerepelnie kell még az `#include <math.h>`-nak is a matematikai függvényekhez. Nem egyes, hanem kis l bet? van a kapcsolóban.
- *-o <program>*: a kimeneti futtatható fájl nevét adja meg
- *<program.c>*: bemeneti forrásfájl nevét adja meg

Kismillió egyéb fordítási opciója is van még a gcc-nek (lásd még *man gcc* és *info gcc*), ezek a tárgy elvégzése szempontjából nem érdekesek.

# 1. gyakorlat (2007-02-13 és 2007-02-16)

A gyakorlat célja, hogy a C nyelv? programozáshoz használandó eszközökkel megismerkedjünk. A bemutatott eszközök: kate, gcc, terminálablak. A gyakorlatnak nem célja megtanítani, hogy a program (pl.

hello100.c) hogyan m?ködik, miért azt csinálja, amit csinál, ezt majd el?adáson tanuljuk meg.

Az szimp2 mappát és a tartalmát csupa kisbet?vel kell létrehozni, vagy ha nagybet?s lett, akkor át kell nevezni kisbet?re.

A Kate nev? szövegszerkeszt?t használjuk.

~/szimp2/hello.c:

```
#include <stdio.h>
int main(void) {
  printf("http://wiki.math.bme.huHello, World!\n"http://wiki.math.bme.hu);
  return 0;
}
```

A sor elején lev? szóközök (ún *indentation*) nem a gép számára, hanem a programot olvasó ember számára hasznosak.

Hasznos parancsok (csak a $ utáni részt kell begépelni, a többit a gép írja ki).

```
$ mkdir ~/szimp2
$ cd ~/szimp2
$ kate hello.c
$ gcc -W -Wall -s -O2 -o hello hello.c
$ ./hello
Hello, World!
```

Windows alatt a kipróbálás úgy történik, hogy Dev-C++-ban lefordítjuk, majd kézzel futtatjuk. A futtatáshoz tehát nem a Dev-C++ *Run* menüpontját választjuk (mert az a futás befejeztével eltünteti az ablakot), hanem kézzel nyitunk egy parancssor-ablakot, a megfelel? cd paranccsal belépünk a megfelel? mappába, majd ott dir paranccsal megnézzünk, milyen `.exe` fájlok vannak (pl. `hello.exe`), és azok egyikét futtatjuk (pl. `hello` paranccsal).

~/szimp2/hello100.c:

```
#include <stdio.h>
int main(void) {
  int c;
  for (c=0;c<100;c++) {
    printf("http://wiki.math.bme.huHello, World!\n"http://wiki.math.bme.hu);
  }
  return 0;
}
```

A sor elején lev? szóközök számára vonatkozó beljebb kezdési szabály (sokféle változata lehetséges, itt csak a tárgy keretében használatosat értelmezzük):

- Beljebb kezdésre csak szóközöket használunk, tabulátort nem.
- A záró kapcsos zárójel sosem kerül egy sor végére, hanem el?tte mindig soremelés van. A soremelés és a záró kapcsos zárójel közé pontosan annyi szóköz kerül, ahány szóköz volt a hozzá tartozó nyitó kapcsos zárójelet tartalmazó sor elején.
- Minden egyéb sor elején pontosan kétszer annyi szóköz van, ahány (bezáratlan) kapcsos zárójelen belül van az adott sor.

A beljebb kezdéshez hasznos tippek:

- A Kate (f?leg *Enter* leütésekor) hajlamos a sor elején 8 egymás utáni szóközöt 1 db tabulátorra cserélni. Ez nekünk nem kívánatos, úgyhogy beszéljük le róla: ikszeljük be a *File / Configure Kate / Editor / Indentation / Use spaces instead of tabs to indent* jelöl?négyzetet.
- Egyes fejleszt?környezetnek van *Smart indent* opciója. Ezt kapcsoljuk át *Autoindentre*. Sajnos Dev-C++-ban csak *Smart indent* van. A *Smart indent* azt csinálja, hogy a soreleji szóközöket igyekszik kváziintelligensen magától meghatározni, pl. ha a sor elején beírunk egy záró kapcsot, magától csökkenti a soreleji szóközök számát. Néha jól, néha rosszul. Inkább kapcsoljuk ki, mint hogy figyelnünk kelljen a hibázásaira, és folyamatosan küzdenünk kelljen vele.

Hasznos parancsok:

```
$ cd ~/szimp2
$ kate hello100.c
$ gcc -W -Wall -s -O2 -o hello100 hello100.c
$ ./hello100
Hello, World!
...
Hello, World!
$ ./hello100 | wc -l
100
$ ./hello100 >hello100.out
$ kate hello100.out
$ ./negyzet100 | more
$ ./negyzet100 | less
```

~/szimp2/negyzet100.c:

```
#include <stdio.h>
int main(void) {
  int c;
  for (c=0;c<=100;c++) {
    printf("http://wiki.math.bme.huszam: %3d, negyzete: %5d\n"http://wiki.math.bme.hu, c, c*c);
  }
  return 0;
}
```

~/szimp2/kettohatvany100.c:

```
#include <stdio.h>
int main(void) {
  int c;
  for (c=0;c<=100;c++) {
    printf("http://wiki.math.bme.huszam: %3d, kettohatvany: %d\n"http://wiki.math.bme.hu, c, 1<<c)
  }
  return 0;
}
```

Az `a<<b` m?velet a-t megszorozza 2 a b-edikennel, és visszaadja az eredményt. Hasonlóa az `a>>b`, de ? szorzás helyett oszt (a 0 felé kerekítve).

Figyeljük meg, hogy 1<<31 már negatív, és 1<<32 == 1. Ezek a túlcsordulás miatt vannak.

# 1. el?adás (2007-02-16)

Megtanultuk:

- bemenet

- kimenet
- processzor
- memória
- program
- bájt: 1 bájt == 8 bit; 0..255
- folyamatábra
- *
- +
- <<
- < <= == > >=
- !=
- a=b=0
- if
- for
- while
- goto
- printf-b?l a %d és a \n
- getchar: while (0<=(c=getchar()))
- putchar
- else: csak érintettük. Példa: if (a<0) b=-1; else b=1;

~/szimp2/bajtszamlalo.c:

```
/* Megszámolja és kiírja, hogy a bemenet hány bájtból áll.
 * `n=n+1' helyett `n++'-ot írtunk.
 */
#include <stdio.h>
int main(void) {
  int n=0;
  while (0<=getchar()) n++;
  printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, n);
  return 0;
}
```

Kipróbálás:

```
$ echo sok | ./bajtszamlalo
4
$ echo -n sok | ./bajtszamlalo
3
```

~/szimp2/sorszamlalo.c:

```
/* Megszámolja és kiírja, hogy a bemenet hány sorból áll.
 * Az utolsó sor nem számít, ha nincs a végén soremelés.
 */
#include <stdio.h>
int main(void) {
  int c, n=0;
  while (0<=(c=getchar())) {
    if (c=='\n') n++;
  }
  printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, n);
  return 0;
}
```

Kipróbálás:

```
$ ls /bin/bash /usr/bin/id /dev/null /etc/inputrc | ./sorszamlalo
4
```

# 2. gyakorlat (2007-02-20)

A gyakorlat megkezdése el?tt ezt az útmutatót mindenki figyelmesen olvassa el, a gyakorlaton pedig figyelmesen kövesse. A figyelmes követésnek része, hogy a bemásolandókat pontosan másoljuk be, a fájlnevet pontosan úgy adjuk meg, ahogy le van írva, minden karakter számít (tehát nem inputrc, hanem ~/.inputrc).

Ugyanazt a parancsot sosem írjuk be kétszer, hanem a korábbi beírást hasznosítjuk újra.

A PageUp-pal és PageDown-nal történ? history-kereséshez az ~/.inputrc fájlukba írjuk bele:

```
set meta-flag on
set input-meta on
set convert-meta off
set output-meta on
"http://wiki.math.bme.hu\e[5~"http://wiki.math.bme.hu: history-search-backward
"http://wiki.math.bme.hu\e[6~"http://wiki.math.bme.hu: history-search-forward
```

Ezután vagy nyissunk új terminálablakot, vagy adjuk ki:

```
$ bind -f ~/.inputrc
```

Az aktuális terminálablakban használhatjuk a *history* parancsot, egy már becsukott terminálablak esetén pedig a ~/.bash_history fájlt. Érdemes e fájlból a legfontosabb parancsokat másik fájlba írni, mert a bash a régi bejegyzéseket eldobja. A limit megnövelhet?, ha a ~/.bashrc-nkbe elhelyezzük az alábbi sort:

```
export HISTSIZE=9999
```

Ha egy programnak többször akarjuk ugyanazt a bemenetet adni, akkor a bemenetet írjuk be fájlba (pl. *prog.in*), és irányítsuk át. Ennek segítségével könnyen ellen?rizhet?, hogy a program két változata (*prog1* és *prog2*) ugyanazt a kimenetet produkálja-e:

```
$ ./prog1 <prog.in >prog1.out
$ ./prog2 <prog.in >prog2.out
$ diff prog1.out prog2.out
```

A diff program kimenetét kell figyelni.

Ha a program bemenetét UNIX-on a terminálon gépeljük be, akkor az alábbiakra figyeljünk:

- Egy egész sort be kell írni (és *Enter*-rel lezárni), és a program csak ezután kapja meg az egész sort.
- A sorvégi *Enter*-t a program `'\n'`-ként kapja.
- A bemenet végét (EOF, -1) *Enter*, majd Ctrl-*D* lenyomásával jelezhetjük. Ezután a program következ? getchar() hívása -1-et ad vissza, a scanf() pedig (általában) nullát.
- Windows alatt Ctrl-*D* helyett *Enter*, Ctrl-*Z*, *Enter*.
- Ha Ctrl-*C*-t nyomunk, az örökre megszakítja a program futását, tehát ha a program kilépés el?tt még kiírt volna valamit, akkor azt már nem fogja kiírni.

Számkitalálós program (embergondol.c: ember gondol, gép talál ki):

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

```
  int a=1, b=100, f, c;
  printf("http://wiki.math.bme.huGondolj egy szamot %d es %d kozott (zart)!\n"http://wiki.math.bme
  while (a!=b) {
    f=(a+b)/2;
    printf("http://wiki.math.bme.huNagyobb, mint %d?\n"http://wiki.math.bme.hu, f);
    while ((c=getchar())=='\n') {}
    if (c=='i') a=f+1;
    else if (c=='n') b=f;
  }
  return 0;
}
```

A program találja ki a gondolt számot. Az ember i-vel vagy n-nel válaszol pl. arra a kérdésre, hogy *A gondolt szám nagyobb, mint 50?*. Az ember válaszát getchar()-ral kel beolvasni.

Az alábbira már nem jutott kedden id?, pedig jóval egyszer?bb, mint az embergondol.

Számkitalálós program (gepgondol.c: gep gondol, ember talál ki):

```
#include <stdio.h>
#include <math.h>
#include <time.h>
int main(void) {
  int a=1, b=100;
  int gondolt, tipp;
  srand(time());
  gondolt=rand()%(b-a)+a;
  printf("http://wiki.math.bme.huGondoltam egy szamot %d es %d kozott (zart), talald ki!\n"http://
  while (1) {
    printf("http://wiki.math.bme.huMit tippelsz?\n"http://wiki.math.bme.hu);
    scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu,&tipp);
    if (gondolt>tipp) {
      printf("http://wiki.math.bme.huA gondolt szam nagyobb, mint %d\n"http://wiki.math.bme.hu, ti
    } else {
      printf("http://wiki.math.bme.huA gondolt szam nem nagyobb, mint %d\n"http://wiki.math.bme.hu
    }
  }
  return 0;
}
```

# 2. el?adás (2007-02-23)

Megtanultuk:

- egydimenziós tömbök definiálása és deklarálása
- [*0..(n-1)*] indexeljük az *n* elem? tömböt
  - tömbelem hozzáférése nagyon gyors (konstans id?ben történik)
  - nagy félrecímzés általában gyors és felt?n?, kicsi félrecímzés alattomos és nehezen felderíthet? hibához vezet
  - gyakorlaton lesz még szó ezek felderítésér?l
- több (pl. kett?) dimenziós tömbök definiálása és deklarálása
- tömbelemek inicializálása (`for`) ciklussal, pl. angol abc, mátrix kinullázása
- hogyan lehet beszúró rendezéssel egy tömböt rendezni
  - költsége $O(n^2)$, viszont nincs járulékos tárhely igénye
  - m?ködik láncolt listákra is
    - az elemhozzáférés tömbnél konstans, listánál O(n) id?igény?
    - az elem beszúrása listánál O(1), tömbnél O(n) id?igény?
  - `#define MAX 0x100` után a fordító `MAX`-ot 0x100-ra fogja mindig kicserélni

    - az algoritmus m?ködésér?l egy szép animáció: http://www.cs.bme.hu/~gsala/alg_anims/3/isort-e.html
- hogyan lehet mátrixot transzponálni
- a "http://wiki.math.bme.hu%6d"http://wiki.math.bme.hu formázó sztringgel a printf a 6-nál nem több számjegy? számokat ?szépen?, jobbra igazítva fogja kiírni
- a 0x100 az hexadecimális, azaz 16-os számrendszerben vett 100, tehát 256

~/szimp2/beszurorendezes.c:

```
/* A bemenetr?l beolvasott számokat beszúró rendezéssel rendezi,
 * majd kiírja a kimenetre.
 */
#include <stdio.h>
#define MAX 0x10
int main(void) {
  int tomb[MAX], elem, i, j;
  for(i = 0; i != MAX; i++)
    scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu, &tomb[i]);
  for (i = 1; i < MAX; i++) {
    elem = tomb[i];
    j = i-1;
    while (j >= 0 && tomb[j] > elem) {
      tomb[j+1] = tomb[j];
      j--;
    }
    tomb[j+1] = elem;
  }
  for(i = 0; i != MAX; i++)
    printf("http://wiki.math.bme.hu%d \n"http://wiki.math.bme.hu, tomb[i]);
  return 0;
}
```

Kipróbálás:

```
$ ./beszurorendezes > rendezett # és beírunk 16 álvéletlen számot
86 36 16 58 2 64 40 7 14 31 50 96 54 18 76 61
$ sort -n | diff rendezett -    # és beírjuk pontosan ugyanazokat a számokat
86 36 16 58 2 64 40 7 14 31 50 96 54 18 76 61
# és itt semmit sem szabad kiírnia, hiszen nincs eltérés a Unixos és az
# általunk írt rendezés eredménye között
```

~/szimp2/matrixtranszponalas.c:

```
#include <stdio.h>
int main(void) {
  int a[10][6], b[6][10], n, m;

  for (n=0; n < 10; n++) {
    printf("http://wiki.math.bme.huIrd be a(z) %d. sort (6 db számot):\n"http://wiki.math.bme.hu,
    for(m=0; m < 6; m++) {
      scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu, &a[n][m]);
    }
  }

  for (n=0; n < 10; n++)
    for(m=0; m < 6; m++)
      b[m][n] = a[n][m];

  for (n=0; n < 6; n++) {
    for(m=0; m < 10; m++) {
      // persze kiirhatnank a[m][n]-et is, es akkor nem is kell transzponalni
```

```
      printf("http://wiki.math.bme.hu%6d "http://wiki.math.bme.hu, b[n][m]);
    }
    printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
  }
  return 0;
}
```

Ez nem volt el?adáson, de érdekes (segédtömb használata nélkül transzponálja a mátrixot):

~/szimp2/matrixtranszponalas_helyben.c:

```
#include <stdio.h>
int main(void) {
  int a[6][6], c;
  unsigned n, m;

  for (n=0; n < 6; n++) {
    printf("http://wiki.math.bme.huIrd be a(z) %d. sort (6 db számot):\n"http://wiki.math.bme.hu,
    for (m=0; m < 6; m++) {
      scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu, &a[n][m]);
    }
  }

  for (n=0; n < 6; n++) {
    for (m=n+1; m < 6; m++) {
      c=a[n][m]; a[n][m]=a[m][n]; a[m][n]=c;
    }
  }

  for (n=0; n < 6; n++) {
    for (m=0; m < 6; m++) {
      printf("http://wiki.math.bme.hu%6d "http://wiki.math.bme.hu, a[n][m]);
    }
    printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
  }
  return 0;
}
```

# 3. gyakorlat (2007-02-27, 2007-03-02)

A feladat az egymilliónál kisebb prímszámok kiírása volt Eratoszthenészi szita segítségével, ~/szimp2/erszita1.c néven.

~/szimp2/erszita1.c:

```
/* Esze Ágnes megoldása alapján */
#include <stdio.h>
int main (void) {
  int i,j;
  char tomb[1000000];
  for(i=2;i<1000000;i++) {
    tomb[i]=0;
  }
  for(i=2;i<1000000;i++) {
    if (tomb[i]!=1){
      printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, i);
      for (j=2*i; j<1000000; j+=i) {
        tomb[j]=1;
      }
    }
```

```
  }
  return 0;
}
```

Kipróbálás:

```
$ cd ~/szimp2
$ gcc -W -Wall -s -O2 -o erszita1 erszita1.c
$ ./erszita1 | wc  -l
78498
$ ./erszita1 | head -5
2
3
5
7
11
$ ./erszita1 | tail -5
999953
999959
999961
999979
999983
```

Ezután javítani kellett a program sebességén úgy, hogy az 1000 fölötti prímek többszöröseit ne húzza ki (tehát nincs `tomb[p]=1`), mert azokat már az 1000 alatti prímek többszöröseként már kihúzta. Ehhez az alábbi módosítások voltak szükségesek:

- Ha a talált prím p, akkor csak p*p-t?l 999999-ig kell kihúzni a többszörösöket.
- Amikor p*p el?ször n? 999999 fölé, kilépünk a for-ciklusból, és egy második for-ciklisban kiírjuk az 1000 fölötti prímeket.

~/szimp2/erszita2.c:

```
/* Esze Ágnes megoldása alapján */
#include <stdio.h>
char tomb[1000000];
int main (void) {
  int i,j;
  for(i=2;i<1000000;i++) {
    tomb[i]=0;
  }
  for(i=2;i<1000000;i++) {
    if (i*i>=1000000)
      goto hoppa;
    if (tomb[i]!=1){
      printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, i);
      for (j=i*i; j<1000000; j+=i) {
        tomb[j]=1;
      }
    }
  }
 hoppa:
  for(;i<1000000;i++) {
    if (tomb[i]!=1)
      printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, i);
  }
  return 0;
}
```

Kipróbálás ugyanúgy, mint erszita1 esetén, de erszita1 helyett erszita2-vel.

Megjegyzés: nagy tömböket (> kb. 8 MB) a függvényen kívülre érdemes rakni, mert különben *Segmentation fault* (Szegmens hiba) lesz.

Harmadik, szorgalmi feladatként azt kellett megoldani, hogy a páros számokkal nem is foglalkozunk (kivéve a 2-t), a páros indexeket a tömbb?l is kihagyjuk. Tehát például a p=13-nak megfelel? hely a tomb[6], a p=17-nek megfelel? hely pedig a tomb[8].

Ehhez `tomb[valami]` helyett elég `tomb[valami/2]`-t kell írni, és kettesével kell ugrálni.

~/szimp2/erszita3.c

```
/* Esze Ágnes megoldásából indulva */
#include <stdio.h>
char tomb[500000];
int main (void) {
  int i,j;
  for(i=0;i<500000;i++) {
    tomb[i]=0;
  }
  printf("http://wiki.math.bme.hu2\n"http://wiki.math.bme.hu);
  for(i=3;i<1000000;i+=2) {
    if (i*i>=1000000)
      goto hoppa;
    if (tomb[i/2]!=1){
      printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, i);
      for (j=i*i; j<1000000; j+=2*i) {
        tomb[j/2]=1;
      }
    }
  }
 hoppa:
  for(;i<1000000;i+=2) {
    if (tomb[i/2]!=1)
      printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, i);
  }
  return 0;
}
```

Érdemes egymás mellett összevetni az erszita2.c-t és az erszita3.c-t.

# 3. el?adás (2007-03-02)

A függvényekr?l és a rekurzióról volt szó. Ezeket a fogalmakat vettük:

- függvény
- visszatérési érték
- paraméter
- lokális változó
- érték szerinti paraméterátadás
- cím szerinti paraméterátadás
- mellékhatás
- rekurzió

Az alábbi megoldás nem jól cserél, mert érték szerint veszi át a-t és b-t, tehát a hívás pillanatában lemásolja, és csak a másolatot cseréli, noha az eredetit kéne:

```
void rosszcsere(int a, int b) {
```

```
  int abak=a;
  a=b;
  b=abak;
}

int main(void) {
  int x=5, y=6;
  printf("http://wiki.math.bme.hucsere elott x=%d, y=%d\n"http://wiki.math.bme.hu, x, y); /* 5, 6
  rosszcsere(&x, &y);
  printf("http://wiki.math.bme.hucsere utan  x=%d, y=%d\n"http://wiki.math.bme.hu, x, y); /* 5, 6
  return 0;
}
```

Példa cím szerinti paraméterátadásra (~/szimp2/csere.c; ez már jó, és szerepelt is a táblán):

```
void csere(int *a, int *b) {
  int abak=*a;
  *a=*b;
  *b=abak;
}

int main(void) {
  int x=5, y=6;
  printf("http://wiki.math.bme.hucsere elott x=%d, y=%d\n"http://wiki.math.bme.hu, x, y);
  csere(&x, &y);
  printf("http://wiki.math.bme.hucsere utan  x=%d, y=%d\n"http://wiki.math.bme.hu, x, y);
  return 0;
}
```

Példa a hatványozás rekurzív számolására (~/szimp2/pow.c):

```
/** Visszadja a az alap**kitevo hatványozás eredményét.
 * Csak akkor m?ködik helyesen, ha kitevo>=0.
 * pow(0,0)==1
 */
int pow(int alap, int kitevo) {
  if (kitevo<=0) return 1;
  return alap*pow(alap, kitevo-1);
}
```

Példa moduláris hatványozás rekurzív számolására (~/szimp2/powmod.c):

```
/** Visszadja a az alap**kitevo hatványozás eredményének modulo modulus vett
 * maradékát.
 * Csak akkor m?ködik helyesen, ha alap>=0 és kitevo>=0 és modulus>=2.
 * powmod(0,0,modulus)==1
 */
int powmod(int alap, int kitevo, int modulus) {
  if (kitevo<=0) return 1;
  return ((alap%modulus)*pow(alap, kitevo-1, modulus))%modulus;
}
```

Példa moduláris hatványozás rekurzív számolására az el?z?nnél gyorsabban (~/szimp2/powmod2.c):

```
/** Visszadja a az alap**kitevo hatványozás eredményének modulo modulus vett
 * maradékát.
 * Csak akkor m?ködik helyesen, ha alap>=0 és kitevo>=0 és modulus>=2.
 * powmod2(0,0,modulus)==1
 */
int powmod2(int alap, int kitevo, int modulus) {
  int ret;
```

```
  if (kitevo<=0) return 1;
  alap%=modulus;
  ret=powmod2(alap, kitevo/2, modulus);
  if (ret!=0) {
    ret=(ret*ret)%modulus;
    if (kitevo%2!=0) ret=(alap*ret)%modulus;
  }
  return ret;
}
```

A fenti powmod2 például 5 a 11-edikent modulo 10 így számolja ki:

```
o=5 % 10;
p=o*o % 10;
q=5*p*p % 10;
r=5*q*q % 10;
return r;
```

Csemegeként (nem kerül számonkérésre) következzen a powmod2 olyan változata, ami a lokális változók értékét nem módosítja (~/szimp2/powmod3.c):

```
/** Visszadja a az alap**kitevo hatványozás eredményének modulo modulus vett
 * maradékát.
 * Csak akkor m?ködik helyesen, ha alap>=0 és kitevo>=0 és modulus>=2.
 * powmod2(0,0,modulus)==1
 */
int powmod3(int alap, int kitevo, int modulus) {
  int ret;
  if (kitevo<=0) return 1;
  return powmod3b(alap%modulus, kitevo, modulus);
}

/** Csak akkor m?ködik helyesen, ha alap>=0 és alap<modulus és modulus>=2 és
 * kitevo>=0.
 */
int powmod3b(int alap, int kitevo, int modulus) {
  return powmod3c(alap, kitevo, modulus, powmod3b(alap, kitevo/2, modulus));
}

/** Csak akkor m?ködik helyesen, ha alap>=0 és alap<modulus és modulus>=2 és
 * kitevo>=0.
 */
int powmod3c(int alap, int kitevo, int modulus, int fele) {
  if (fele==0) return 0;
  return powmod3d(alap, kitvo, modulus, (fele*fele)%modulus);
}

/** Csak akkor m?ködik helyesen, ha alap>=0 és alap<modulus és modulus>=2 és
 * kitevo>=0.
 */
int powmod3d(int alap, int kitevo, int modulus, int felnegyzet) {
  if (kitevo%2==0) return felnegyzet;
  return (alap*felnegyzet)%modulus;
}
```

Megjegyzés: a powmod3 függvény Cékla nyelven is m?ködik. A Cékla nyelvet majd a Deklaratív programozás tárgy keretében fogjuk tanulni.

Példa a faktoriális rekurzív számolására (~/szimp2/fakt.c):

```
int fakt(int n) {
```

```
  if (n<2) return 1;
  return n*fakt(n-1);
}
```

Példa a faktoriális rekurzív számolására akkumulátorral és jobbrekurzióval (~/szimp2/fakt2.c; nem kell érteni, hogy miért jobb a fakt2, mint a fakt):

```
int fakt2(int n) {
  return fakt2b(n, 1);
}

/** n!*szorzo -t adja vissza. */
int fakt2b(int n, int szorzo) {
  if (n<2) return szorzo;
  return fakt2b(n-1, szorzo*n);
}
```

Példa a Fibonacci-sorozat rekurzív számolására (~/szimp2/fib.c; ez szerepelt is a táblán el?adáson):

```
int fib(int n) {
  if (n<2) return n;
  return fib(n-1)+fib(n-2);
}
```

Miért olyan lassú fib(1000)-t kiszámolni? Azért, mert fib(1000)-hez pl. fib(1)-et és fib(2)-t rengetegszer kell kiszámolni, noha elég lenne egyszer is. Ilyenkor az iteratív (értsd: for-ciklusos) megoldás jóval gyorsabb:

Példa a Fibonacci-sorozat iteratív számolására (~/szimp2/fib2.c; ez is szerepelt a táblán):

```
int fib(int n) {
  int a, b, regib;
  if (n<2) return n;
  a=0; b=1;
  while (n>1) {
    regib=b; b+=a; a=regib;
    n--;
  }
  return b;
}
```

Példa a legrövidebb közös részstring hosszának rekurzív meghatározására (~/szimp2/lkrh.c). Ezt még nem kell érteni.

```
int max(int a, int b) {
  return a>b ? a : b;
}

int lkrh(char *s, char *t) {
  if (*s=='\0') return 0; /* ha az s üres, akkor lkrh==0 */
  if (*s=='\0') return 0; /* ha a  t üres, akkor lkrh==0 */
  if (*s==*t) { /* az els? karakter azonos */
    return 1+lkrh(s+1, t+1);
  } else {
    return max(lkrh(s+1, t), lkrh(s, t+1));
  }
}
```

A fenti lkrh függvény túl lassú, mert rekurzív hívásakor ugyanarra az s--t párra többször is meghívódik (és tovább hívja magát rekurzívan). Jó lenne az els? hívás után letárolni az eredményt.

# 4. gyakorlat (2007-03-06, 2007-03-09)

Ezen a gyakorlaton a függvények használatát gyakorljuk:

- a kód egyszer?sítése az ismétl?d? feladatok függvénybe rakásával
- érték szerinti paraméterátadás
- rekurzió
- érték módosítása cím szerinti paraméterátadással
- több érték visszadása cím szerinti paraméterátadással

Törtszámösszeadó programot fogunk írni. Minden tört egy egész érték? számálálóból és egy nem nulla egész érték? nevez?b?l áll. A tört értéke a számláló és a nevez? hányadosa. Egy tört egyszer?sítve van, ha a nevez?je pozitív, és a számláló és a nevez? legnagyobb közös osztója 1. Két tört összeadása úgy történik, hogy el?ször egyszer?sítjük ?ket (külön-külön); majd képezzük a két nevez? legkisebb közös többszörösét; b?vítjük mindkét törtét úgy, hogy a közös nevez?jük a legkisebb közös többszörös legyen; a b?vített számlálókat összeadjuk; az eredményt egyszer?sítjük.

A törtszámösszeadó program a bemenetr?l beolvas négy egész számot: *as*, *an*, *bs*, *bn*, majd kiszámolja az *as/an' és* bs/bn *törtek összegét, és perjellel elválasztva kiírja az eredmény számlálóját (*es*) és nevez?jét (*en*) egy sorba. Ezután a következ? feladványsor beolvasásával folytatja.*

Példa bemenet (~/szimp2/tortad1.in):

```
1 2 3 4
3 4 5 -6
3 4 -5 6
-3 4 5 6
3 6 30 20
```

Példa kimenet (~/szimp2/tortad1.exp):

```
5/4
-1/12
-1/12
1/12
2/1
```

A megírandó program vázlata (~/szimp2/tortad.c):

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

/** Az a és b pozitív egész számok legnagyobb közös osztóját adja vissza.
 * Tilos negatív számmal hívni.
 */
int lnko(int a, int b) {
  if (a<0 || b<0) abort();
  ... /* Euklideszi algoritmus: ha a==0, akkor b, egyébként ... rekurzió */
}

/** Az a és b pozitív egész számok legkisebb közös többszörösét adja vissza.
 * Tilos 0-val vagy negatív számmal hívni.
 */
int lkkt(int a, int b) {
  if (a<1 || b<1) abort();
  ... /* használjuk az lnko()-t */
}
```

```
/** Az s/n törtet egyszer?síti. A negatívságot felviszi a számlálóba, és
 * mindkét számot leosztja a legnagyobb közös osztójukkal. Nulla nevez?vel
 * tilos meghívni.
 */
void egyszerusit(int *s, int *n) {
  if (*n==0) abort();
  ...
}

/** Az as/an törthöz hozzáadja a bs/bn törtet, és az eredményt az es/en
 * törtben adja vissza.
 */
void osszead(int as, int an, int bs, int bn, int *es, int *en) {
  ... /* egyszer?sít, közös nevez?re hoz, összead, egyszer?sít */
}

int main(void) {
  int as, an, bs, bn, es, en;
  while (4==scanf("http://wiki.math.bme.hu%d%d%d%d"http://wiki.math.bme.hu, &as, &an, &bs, &bn)) {
    ... /* az osszead() függvényt kell megfelel? paraméterekkel meghívni */
    ... printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, es); /* en-t is ki kell még í
  }
  return 0;
}
```

A kész program (~/szimp2/tortad.c):

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

/** Az a és b pozitív egész számok legnagyobb közös osztóját adja vissza.
 * Tilos negatív számmal hívni.
 */
int lnko(int a, int b) {
  if (a<0 || b<0) abort();
  if (a==0) return b;
  return lnko(b%a, a);
}

/** Az a és b pozitív egész számok legkisebb közös többszörösét adja vissza.
 * Tilos 0-val vagy negatív számmal hívni.
 */
int lkkt(int a, int b) {
  if (a<1 || b<1) abort();
  return a/lnko(a,b)*b;
  /* ^^^ el?ször osztunk, utána szorzunk, hogy ne legyen túlcsordulás */
}

/** Az s/n törtet egyszer?síti. A negatívságot felviszi a számlálóba, és
 * mindkét számot leosztja a legnagyobb közös osztójukkal. Nulla nevez?vel
 * tilos meghívni.
 */
void egyszerusit(int *s, int *n) {
  int c;
  if (*n==0) abort();
  if (*n<0) {
    *n=-*n;
    *s=-*s;
  }
  if (*s<0) {
    c=lnko(-*s,*n);
  } else {
    c=lnko(*s,*n);
```

```
  }
  /* a c-t jobb el?re kiszámolni, mert ha lent c helyére lkno(*s,*n) kerülne,
   * akkor a második lnko(*s,*n) már a módosított s-sel számolna, ami nem jó
   */
  *s/=c;
  *n/=c;
  /* nincs return, mert void a visszatérési érték */
}

/** Az as/an törthöz hozzáadja a bs/bn törtet, és az eredményt az es/en
 * törtben adja vissza.
 */
void osszead(int as, int an, int bs, int bn, int *es, int *en) {
  egyszerusit(&as,&an);
  egyszerusit(&bs,&bn);
  *en=lkkt(an,bn);
  *es=(*en/an)*as+(*en/bn)*bs;
  egyszerusit(&*es,&*en);
  /* ^^^ az & az egyszerusit() deklarációjában lev? * miatt kell */
  /* ^^^ a  * az osszead()     deklarációjában lev? * miatt kell */
  /* ^^^ megjegyzés: &* kihagyható, csak didaktikai okokból van bent */
}

int main(void) {
  int as, an, bs, bn, es, en;
  while (4==scanf("http://wiki.math.bme.hu%d%d%d%d"http://wiki.math.bme.hu, &as, &an, &bs, &bn)) {
    osszead(as, an, bs, bn, &es, &en);
    printf("http://wiki.math.bme.hu%d/%d\n"http://wiki.math.bme.hu, es, en);
  }
  return 0;
}
```

A kipróbálás el?tt készítsük el a mintabemenetet (~/szimp2/tortad1.in) és a mintakimenetet (~/szimp2/tortad1.exp), lásd fent. Ezután kipróbálható:

```
$ cd ~/szim2p
$ gcc -W -Wall -s -O2 -o tortad tortad.c
$ ./tortad <tortad1.in
5/4
-1/12
-1/12
1/12
2/1
$ ./tortad <tortad1.in >tortad1.out
$ diff tortad1.exp tortad1.out
$ _
```

# 4. el?adás (2007-03-09)

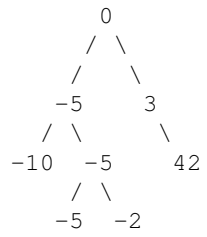
Vettük a *bináris fa* definícióját, az alábbi fogalmakkal:

- csúcs
- él
- gyerek
- szül?
- bináris fa (ha minden csúcsnak 0, 1 vagy 2 gyereke van)
- gyökér
- bels? csúcs (akinek van gyereke -- akár a gyökér is lehet bels? csúcs)
- levél (akinek nincs gyereke -- akár a gyökér is lehet levél)
- bal részfa

- jobb részfa

Vettük a bináris keres?fát. A bináris keres?fa olyan bináris fa, melynek minden csúcsában egy érték szerepel, és minden csúcsára igaz, hogy a bal részfában szerepl? értékek mind legfeljebb akkorák, mint a csúcsban szerepl? érték, és a jobb részfában szerepl? értékek mind legalább akkorák, mint a csúcsban szerepl? érték. Néha kikötik, hogy egy érték csak egyszer szerepelhet (ez egyszer?síti a bináris fában végzett m?veleteket).

Példa bináris keres?fára (az órán is ez volt a táblán):



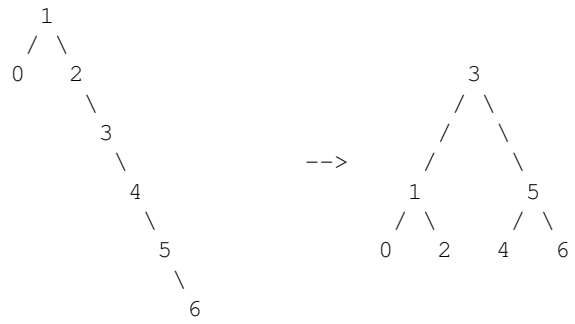

Alapm?veletek bináris keres?fákkal:

- keresés
- új érték beszúrása
- érték törlése

Egyéb m?velet bináris keres?fákkal:

- kiegyensúlyozás: a fa átalakítása a benne lev? értékek megtartásával úgy, hogy a magassága (jóval) kisebb legyen.

Példa kiegyensúlyozásra:




Kiegyensúlyozott fa (ez egy nem túl precíz fogalom): olyan bináris fa, amelynek minden csúcsára igaz, hogy a bal részfában nagyjából ugyanannyi csúcs van, mint a jobb részfában.

Részfa magassága: a részfa gyökerét?l egyik leveléig men? leghosszabb úton lev? csúcsok száma. (A definícióból következik, hogy egy levél magassága 1.)

AVL-fa (precíz fogalom, a kiegyensúlyozott fák egy fajtája): olyan bináris fa, melynek minden csúcsára igaz, hogy a bal részfa magassága és a jobb részfa magassága közti különbség -1, 0 vagy +1.

Keresés bináris keres?fában: A feladat az, hogy keressünk egy olyan csúcsot a fában, ahol az adott érték szerepel. Ha több ilyen csúcs is van, akkor bármelyik jó.

A bináris fában való keresés (rekurzív) algoritmusa:

1. A gyökért?l indulunk.
2. Ha az aktuális csúcsban a kívánt érték szerepel, készen vagyunk.
3. Ha a bal részfa nem üres, és az aktuális csúcsban a kívánt értéknél nagyobb érték szerepel, a bal részfa gyökerével folytatjuk (a 2. lépést?l).
4. Ha a jobb részfa nem üres, és az aktuális csúcsban a kívánt értéknél kisebb érték szerepel, a jobb részfa gyökerével folytatjuk (a 2. lépést?l).
5. A keresést befejezzük, az adott érték nem található meg a fában.

Beszúrás bináris keres?fába: A feladat az, hogy szúrjunk be a fába egy új értéket úgy, hogy az továbbra is bináris keres?fa maradjon.

Látható, hogy a keresés legfeljebb annyi lépésb?l áll, amennyi a fa magassága. AVL-fánál (és sok egyéb kiegyensúlyozott fánál is) egy $n$ csúcsot tartalmazó fa magassága $O(\log n)$, tehát a keresés gyors. Széls?séges esetben, például ha a fa csúcsainak csak jobboldali gyerekük van, az egész fát be kellhet járni ($O(n)$).

C nyelvben a bináris fák kezeléséhez struktúrákat használunk. A kapcsolódó fogalmak:

- struktúra (struct)
- mez?
- struktúra mérete
- mez? kezd?címe a struktúrában
- bájtra igazítás (alignment)

PC (i386, x86) architektúrán az igazítások:

- char esetén 1
- short esetén 2
- double esetén 8
- minden más (pl. int, long, float, struktúra és mutató) esetben 4

Példa struktúra definiálására:

```
struct pelda {
  int a, b;
  char c;
  double d;
};
```

A példa struktúra mérete igazítás nélkül 4+4+1+8 bájt lenne, igazítással viszont 4+4+1+3+8 bájt, mivel a double típusú változónak 4-gyel osztható címen kell kezd?dnie, így 4+4+1 helyett a 4+4+1+3 címen kezd?dik.

Példa struktúra használatára:

```
struct pelda egyik, masik;
egyik.a=5; egyik.b=6; egyik.c='x', egyik.d=egyik.a/2.0;
masik=egyik; /* minden mez?t lemásol */
masik.a=egyik.b; masik.b=egyik.a;
```

Vettük még el?adáson a mutató (pointer) fogalmát. A mutató egy adott típusú érték kezd?címét tartalmazza. Mutató deklarálásához a változónév elé tegyünk egy csillagot:

- `int p;`: a p változó egy int értéket tartalmaz
- `int *p;`: a p változó egy int értékre mutató mutatót tartalmaz

- `int **p;`: a p változó egy int értékre mutató mutatóra mutató mutatót tartalmaz
- `char **argv;`: az argv változó egy char értékre mutató mutatóra mutató mutatót tartalmaz
- `struct pelda *oda;`: az oda változó egy pelda típusnev? struktúrára mutató mutatót tartalmaz

Egy adott változó kezd?címéhez a változónév elé tegyünk egy `&` jelet (ésjelet):

- `int a, *p; p=&a;`: a p mutató az a (int típusú) változó kezd?címét tartalmazza
- `int t[5], *p; p=&t[3];`: a p mutató a t (int típusú elemekb?l álló) tömb 3-adik elemédik kezd?címét tartalmazza
- `struct pelda egyik; int *p; p=&egyik.b;`; a p mutató az *egyik* (pelda típusnev?) struktúra *b* mez?jének kezd?címét tartalmazza

A fordított m?velethez, tehát egy mutató által mutatott memóriaterület értékéhez a mutató elé tegyünk csillagot:

- `int a=5, *p; *p=&a; a=20; *p+=22;`
  `printf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu, *p);`: az a-t el?ször 5-r?l 20-ra változtatja, majd megnöveli 22-vel, majd kiírja (a 42-t)
- `struct pelda egyik; char *p=&egyik.c; *p=5;`: az *egyik* (pelda típusnev?) struktúra *c* mez?jének értékét 5-re változtatja

Speciális mutató a NULL (`#include <stdlib.h>` kell neki), ami nem mutat érvényes memóriaterületre. Példa használatára:

- `int a, *p=NULL; a=*p;`: a második értékadás *Segmentation fault*-ot (Szegmens hiba) okoz, a program futása megszakad. Megjegyzés: *Segmentation fault* más programokban máshogy is el?állhat.

Vegyük észre, hogy a függvényeknél megtanult cím szerinti paraméterátadásnál mutató átadása történik (pl. a törtösszeadós példában `egyszerusit(&as,&es);`), és a függvényen belül a kód csak a mutatót követi (pl. `*s/=c;`).

A bináris fa egy csúcsát struktúraként definiáljuk:

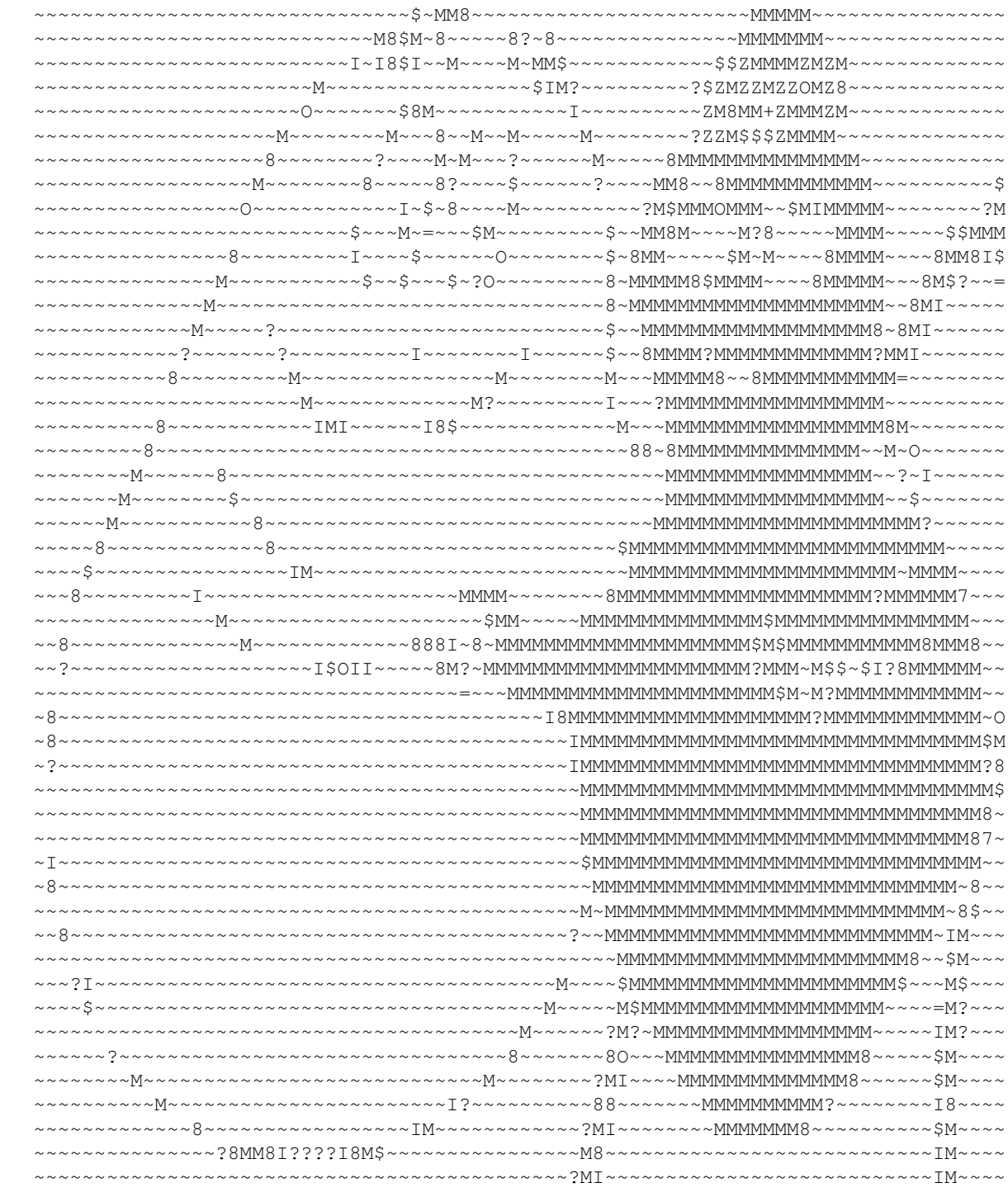
```
struct csucs {
  int ertek;
  struct csucs *bal;
  struct csucs *jobb;
};
```

Példa a fenti bináris fa azon részfájának létrehozására, ami a 3-ból indul lefele (~/szimp2/fapelda1.p):

```
#include <stdlib.h> /* a NULL miatt */

int main(int argc, char**argv) {
  struct csucs a, b;
  a.ertek=3;
  a.bal=NULL;
  a.jobb=&b;
  b.ertek=42;
  b.bal=NULL;
  b.jobb=NULL;
  return 0;
}
```

# 5. gyakorlat (2007-03-10, 2007-03-13)



A mutatók és struktúrák használatát gyakoroljuk. Bináris keres?fák majd csak a következ? gyakorlaton lesznek.

A feladat: Az óvódai évbúcsúztatón kiszámolással döntik el, hogy a csoportból ki lesz az két gyerek, aki Barbapapa[1] elménytábor-bérletet nyer. A csoport tornasorba áll, a két széle összekapcsolódik, majd a legmagasabbtól csökken? sorrendben elmondanak egy kiszámolós mondókát. Akire az utolsó szótag jut, kiesik, majd a kiszámoló a következ? gyerekt?l újraindul mindaddig, amíg több, mint 2 gyerek van. A feladat meghatározni, hogy a gyerekek milyen sorrendben esnek ki, és melyik két gyerek marad bent utoljára.

A gyerekeket tornasorban 1-t?l N-ig számozzuk (az els? a legmagasabb és az N-edik a legalacsonyabb), a mondóka K szótagból áll. 10000 >= N >= 2 és 10000000 >= K >= 1. A bemenet minden sorában N és K található. Minden bemeneti sorhoz egy kimeneti sort kell el?állítani: az sor elejére a kies? gyerekek sorszáma kerül (a kiesés sorrendjében) szóközzel elválasztva, majd egy szóköz, majd egy pontosvessz?, majd egy szóköz, majd a bennmaradó két gyerek sorszáma szóközzel elválasztva (el?ször az utolsó kies?re rákövetkez?, majd az ?rá következ?).

A megoldásban minden gyerekhez fel kell venni egy struktúrát (benne a gyerek sorszámával és a következ? gyerekre mutató mutatóval).

Példa bemenet:

```
9 1
9 2
9 1000
8 10000000
```

A példa bemenethez tartozó kimenet:

```
1 2 3 4 5 6 7 ; 8 9
2 4 6 8 1 5 9 ; 3 7
1 9 7 4 3 2 5 ; 6 8
8 3 7 6 5 1 ; 2 4
```

Az alábbi vázlatból lehet kiindulni (~/szimp2/gyerekki.c):

```
#include <stdio.h>

struct gyerek {
  int szam;
  struct gyerek *kov;
};

struct gyerek t[10000];

int main(void) {
  int n, k;
  int i;
  /** Az akt változó minden kiszámolás kezdete el?tt arra a gyerekre mutat, aki
   * _után_ a kiszámolás elkezd?dik. A kiszámolás végeztével pedig arra a
   * gyerekre mutat, aki _után_ a kiszámolás utolsó szótagja esik.
   */
  struct gyerek *akt;
  while (2==scanf("http://wiki.math.bme.hu%d%d"http://wiki.math.bme.hu, &n, &k)) {
    for (i=0;i<n;i++) {
      ...
    }
    t[n-1].kov=&t[0];
    akt=...;
    ...
    printf("http://wiki.math.bme.hu; %d %d\n"http://wiki.math.bme.hu, ...egyikutolso, ...masikutol
  }
  return 0;
}
```

A kész program (~/szimp2/gyerekki.c):

```
#include <stdio.h>

struct gyerek {
```

```
  int szam;
  struct gyerek *kov;
};

struct gyerek t[10000];

int main(void) {
  int n, k, i, j;
  struct gyerek *akt;
  while (2==scanf("http://wiki.math.bme.hu%d%d"http://wiki.math.bme.hu, &n, &k)) {
    for (i=0;i<n;i++) {
      t[i].szam=i+1;
      t[i].kov=&t[i+1];
    }
    t[n-1].kov=&t[0];
    akt=&t[n-1];
    for (i=0;i+2<n;i++) { /* minden kies? gyerekre */
      for (j=0;j<k-1;j++) akt=(*akt).kov;
      printf("http://wiki.math.bme.hu%d "http://wiki.math.bme.hu, (*(*akt).kov).szam);
      (*akt).kov=(*(*akt).kov).kov;
    }
    printf("http://wiki.math.bme.hu; %d %d\n"http://wiki.math.bme.hu, (*(*akt).kov).szam, (*(*(*ak
  }
  return 0;
}
```

```
#include <stdio.h>

struct gyerek {
  int szam;
  struct gyerek *kov;
};

struct gyerek t[10000];

int main(void) {
  ...
  return 0;
}
```

# 5. el?adás (2007-03-10)

A bináris keres?fák megismerését folytatjuk. Célunk, hogy meg tudjuk írni a bináris keres?fába való beszúrás algoritmusát.

A bináris keres?fába való beszúrás (rekurzív) algoritmusa:

1. A gyökért?l, pontosabban a gyökér kezd?címét?l indulunk.
2. Paraméterként kapjuk az aktuálisan vizsgálandó részfa gyökerének címét. (Üres fa esetén van lényeges különbség: az üres fa gyökere ugyanis NULL, az üres fa gyökerének címe pedig az a memóriaterület, ahol a NULL-t majd átírjuk másra.)
3. Ha az aktuális gyökér NULL, akkor cseréljük le a beszúrandó csúcsra: a csúcs értéke a beszúrandó érték legyen, a bal és jobb mutatók pedig NULL-ok legyenek.
4. Ha a beszúrandó érték megegyezik az aktuális gyökérben található értékkel, és az aktuális gyökérnek nincs bal gyereke, folytassuk a jobb részfával (a 3. lépést?l). (Az algoritmus e lépés nélkül is jól m?ködne.)

5. Ha a beszúrandó érték legfeljebb akkora, mint az aktuális gyökérben található érték, akkor folytassuk a bal részfával (a 3. lépést?l).
6. Folytassuk a jobb részfával (a 3. lépést?l).

Programban:

```
#include <stdlib.h> /* a NULL miatt */

struct csucs {
  int ertek;
  struct csucs *bal;
  struct csucs *jobb;
};

/** Beszúrja a gyoker részfába az uj csúcsot levélként (és egyben NULL-t tesz
 * az uj csúcs bal és jobb mutatóiba.
 */
void beszur(struct csucs **gyoker, struct csucs *uj) {
  while (*gyoker!=NULL) {
    if ((*uj).ertek<=(**gyoker).ertek) {
      gyoker=&(**gyoker).bal;
    } else {
      gyoker=&(**gyoker).jobb;
    }
  }
  *gyoker=uj;
  (*uj).bal=NULL;
  (*uj).jobb=NULL;
}

int main(void) {
  struct csucs *gyoker=NULL;
  struct csucs na, nb, nc, nd, ne, nf, ng, nh;
  na.ertek=  0; beszur(&gyoker, &na);
  nb.ertek= -5; beszur(&gyoker, &nb);
  nc.ertek=-10; beszur(&gyoker, &nc);
  nd.ertek= -5; beszur(&gyoker, &nd);
  ne.ertek= -5; beszur(&gyoker, &ne);
  nf.ertek= -2; beszur(&gyoker, &nf);
  ng.ertek=  3; beszur(&gyoker, &ng);
  nh.ertek= 42; beszur(&gyoker, &nh);
  return 0;
}
```

Vegyük észre, hogy ha más sorrendben szúrjuk be a csúcsokat, más fát kapunk.

Dupla el?adást tartottunk. Volt még szó az alábbiakról:

- operátor-precedencia
  - a leger?sebbek a postfix unáris operátorok
  - utánuk a prefix unárisak jönnek
  - végül a binárisak, megfelel? sorrendben (lásd a *Standard C reference*-ben)
  - azonos szinten belül az asszociativitás dönti el a sorrendet (balról jobbra vagy jobbról balra)
- ++a és a++
- --a és a--
- (a++)+(++b)
- `(*sor).kov`-be azért kell zárójel, mert a `.` er?sebb, mint a `*`
- *p++ === *(p++)
- (*p)++

Volt egy tömbös-mutatós példa is: egy mutatót növelgettünk egy tömbön belül, és a mutató segítségével a tömb elemeit változtatgattuk.

# 6. gyakorlat (2007-03-20, 2007-03-23)

Egyel?re hagyjuk a bináris fákat. A gyakorlat témája a szélességi keresés.

A bioháború nyitányaként az Ellenség egy szigetvilág ellen gyorslopakodású botsáskákat kíván bevetni. Minden szigetre le fognak dobni egy sáskarajt, amely egyetlen éjszaka alatt elpusztítja a sziget teljes növényvilágát. Az ökoszisztéma hamarosan összeomlik, és a bioháború az Ellenség gy?zelmével ér véget. A bombázást egy repül? végzi, amely nyugatról keletre halad, majd ha végzett egy sávval, akkor egy sávval délebbre visszafelé (keletr?l nyugatra) repül végig, majd megint egy sávot délre lép, és újra nyugatról keletre repül. Ha olyan sziget fölé érkezik, ahová még nem dobott, akkor azonnal ledob egy sáskarajt. A feladat a szigetvilág térképe alapján megszámolni a szigeteket, és meghatározni, hogy az adott szigeten belül hová kell ledobni a sáskarajt.

A szigetvilág négyzetrácsos térkép formájában adott. A # karakter jelöli a szárazföldet, és a . karakter a vizet. A térkép fölött a térkép W szélessége (legfeljebb 1000) és H magassága (legfeljebb 1000) áll. Két szárazföldi négyzet akkor tartozik ugyanahhoz a szigethez, ha vagy élben, vagy csúcsban érintkeznek. Példa térkép:

```
6 5
#.....
..#.#.
##...#
...#..
..#.##
```

A programnak a repül? útját nyomon követve azt kell meghatároznia, hogy az egyes ledobásokig a bombázás kezdete óta mennyi távolságot tett meg a repül?. Például a fenti bemenetre

```
0 7 9 20
```

a válasz, mivel összesen 0 távolságot tett meg a repül? az els? (1 méret?) sziget bombázásáig, összesen 7 távolságot tett meg (ebb?l 1-et dél fele, és 1-et keletr?l nyugatra) a második (2 méret?) sziget bombázásáig, összesen 9 távolságot tett meg a harmadik (3 méret?) sziget bombázásáig, és összesen 20 távolságot tett meg a negyedik (4 méret?) sziget bombázásáig.

A megírandó program váza (~/szimp2/bombaz.c):

```
#include <stdio.h>

char t[1000][1000];
int w, h;

/** Elsüllyeszti t-ben az (x,y) négyzetet és a hozzá csatlakozó szigetet. */
void elsullyeszt(int x, int y) {
  ... /* ha lementünk a térképr?l, vége */
  ... /* ha nem szárazföld van ott, vége */
  ... /* szárazföld átváltoztatása vízzé */
  ... /* 8 rekurzív hívás a szomszédokra */
}

int main(void) {
  int x, y, d, c;
  while (2==scanf("http://wiki.math.bme.hu%d%d"http://wiki.math.bme.hu,&w,&h)) {
```

```
    /* a térkép beolvasása */
    for (y=0;y<h;y++) {
      for (x...) {
        while ((c=getchar()=='\n') {} ... zárójelhiba van a sorban
        ...
      }
    }

    /* a bombázás */
    d=0;
    for (y=0;y<h;y++) {
      for (x...) { /* nyugatról keletre repülünk */
        ...
      }
      if (++y==h) ...
      for (x...) { /* keletr?l nyugatra repülünk */
        ...
      }
    }
  }
}
```

Mintamegoldás (~/szimp2/bombaz.c):

```
#include <stdio.h>

char t[1000][1000];
int w, h;

/** Elsüllyeszti t-ben az (x,y) négyzetet és a hozzá csatlakozó szigetet. */
void elsullyeszt(int x, int y) {
  if (x<0 || y<0 || y>=h || x>=w) return; /* ha lementünk a térképr?l, vége */
  if (t[x][y]=='.') return; /* ha nem szárazföld van ott, vége */
  t[x][y]='.'; /* szárazföld átváltoztatása vízzé */
  /* vvv 8 rekurzív hívás a szomszédokra */
  elsullyeszt(x-1,y-1);
  elsullyeszt(x  ,y-1);
  elsullyeszt(x+1,y-1);
  elsullyeszt(x-1,y);
  elsullyeszt(x  ,y);
  elsullyeszt(x+1,y);
  elsullyeszt(x-1,y+1);
  elsullyeszt(x  ,y+1);
  elsullyeszt(x+1,y+1);
}

int main(void) {
  int x, y, d, c;
  while (2==scanf("http://wiki.math.bme.hu%d%d"http://wiki.math.bme.hu,&w,&h)) {

    /* a térkép beolvasása */
    for (y=0;y<h;y++) {
      for (x=0;x<w;x++) {
        while ((c=getchar())=='\n') {}
        t[x][y]=c;
      }
    }

    /* a bombázás */
    d=0;
    for (y=0;y<h;y++) {
      for (x=0;x<w;x++) { /* nyugatról keletre repülünk */
        if (t[x][y]!='.') {
```

```
          printf("http://wiki.math.bme.hu%d "http://wiki.math.bme.hu, d);
          elsullyeszt(x,y);
        }
        d++;
      }
      if (++y==h) break;
      for (x=w-1;x>=0;x--) { /* keletr?l nyugatra repülünk */
        if (t[x][y]!='.') {
          printf("http://wiki.math.bme.hu%d "http://wiki.math.bme.hu, d);
          elsullyeszt(x,y);
        }
        d++;
      }
    }
    printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
  }
  return 0;
}
```

# 6. el?adás (2007-03-23)

Az el?adáson a struktúrák és mutatók használatát ismételjük, továbbá a mutató-aritmetikát (pl. `*p++=*s++;`) ismerjük meg részletesebben. Újabb példát kapunk egy rekurzív függvényre, melynek egyes példányai ugyanazt a tömböt használják. Megtanuljuk azt is, hogy egy tömb mutatóként átadható a függvénynek.

Az volt a feladat, hogy intervallumok listáját az intervallum fels? (*b*) végpontja szerint rendezzük, összefésül? rendezéssel.

A táblára az alábbi program került:

```
#include <stdio.h>

struct iv {
  int a, b;
};

/** Összefésüli a p-vel kezd?d? ph hosszú rendezett listát a
 * q-val kezd?d?, qh hosszú rendezett listával, és az eredményül
 * kapott ph+qh hosszúságú rendezett listát az r-rel kezd?d?
 * tömbbe helyezi.
 */
void osszefesul(struct iv *p, int ph, struct iv *q, int qh,
 struct iv *r) {
  while (ph>0 && qh>0) {
    if ((*p).b<(*q).b) {
      *r++=*p++; ph--;
    } else {
      *r++=*q++; qh--;
    }
  }
  while (ph>0) { *r++=*p++; ph--; }
  while (qh>0) { *r++=*q++; qh--; }
}

/** Rendezi (a b mez? szerint növekv? sorrendbe) a ph hosszú p
 * tömböt a legalább ph hosszú s segédtömb felhasználásával.
 */
void osszefesulve_rendez(struct iv *p, int ph, struct iv *s) {
  int px, qx;
  struct iv *q;
```

```
  if (ph<2) return;
  px=ph/2;
  qx=ph-px;
  q=p+px;
  osszefesulve_rendez(p, px, s); /* els? fél rendezése */
  osszefesulve_rendez(q, qx, s); /* második fél rendezése */
  osszefesul(p, px, q, qx, s); /* rendezett felek összefésülése */
  while (ph>0) { *p++=*s++; ph--; } /* visszamásolás a segédtömbb?l */
}

int n; /* a t tömbben lev? elemek száma */
struct iv t[1<<23], /* 64 MB */
          s[1<<23]; /* 64 MB */

int main(void) {
  int i;
  while (1==scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu,&n)) {
    for (i=0;i<n;i++) scanf("http://wiki.math.bme.hu%d,%d"http://wiki.math.bme.hu, &t[i].a, &t[i].
    osszefesulve_rendez(t, n, s);
    for (i=0;i<n;i++) printf("http://wiki.math.bme.hu%d,%d "http://wiki.math.bme.hu, t[i].a, t[i].
    printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
  }
  return 0;
}
```

Minta bemenet:

```
5 7,6 5,2 1,0 9,4 3,8
```

Minta kimenet:

```
1,0 5,2 9,4 7,6 3,8
```

Ellen?rz? kérdések:

- Mekkora a `struct iv` mérete? Válasz: mivel az `int` mérete 4 bájt, és 2 db `int` van benne, ezért 8 bájt. Az igazítás ezt a méretet nem növeli, mert az `int`-eket elég 4-gyel osztható pozícióra igazítani.
- Miért van szükség az s segédtömbre? Válasz: Mert az *osszefeseul* függvény nem tudja visszatenni az eredeti tömb(ök)be az eredményt.
- Nem lesz-e abból baj, hogy az *s* segédtömböt az *osszefesulve_rendez* függvény egyes futó példányai egyszerre használják, és egymás adatait felülírják? Válasz: Nem lesz baj, mert az *s* segédtömböt csak az *osszefesulve_rendez* függvény utolsó két sora használja (értsd: írja vagy olvassa), és ebben a két sorban nem történik rekurzív hívás.
- Milyen értéket kap az `osszefesulve_rendez(t, n, s);` hívásban a függvény *p* paramétere? Válasz: A *t* tömb legels? elemének címét kapja, tehát a `t` ugyanazt jelenti, mint `&t[0]`.

# 7. gyakorlat (2007-03-27, 2007-03-30)

Íme a beígért bináris fás gyakorlat. Ámbár nem bináris keres?fás.

Az Egyesült Boldogságbank és Örömosztó Betéti Társaság az ügyféligények fürgébb kielégítésére telefonos ügyintézési rendszert vezet be. Sajnálatos módon -- és emiatt a boldogulni vágyó ügyfelekt?l kell? szánakozással elnézést is kértek -- a gazdasági fejlettség jelen fokán anyagilag nem engedhet? meg, hogy a telefont hús-vér ügyintéz? vegye fel, ezért az agázatban már jól ismert megoldás, a tone üzemmódú telefonos menürendszer mellett döntöttek. A Jóltervez? Osztály már ki is ötlötte az elérhet? menüpontokat, és ki is alakította a menüt, ám ez utóbbit az Elégedettségi Ellen?rz?csoport nem hagyta jóvá arra hivatkozva, hogy a

menürendszer túl bonyolult, és mire az ügyfél el tud jutni a kívánt menüpontig, a maradék jókedve is elillan. Az ellen?rz?csoport az alábbi két irányelvet léptette életbe: 1. minden döntési helyzetben az ügyfélnek pontosan 2 lehet?ség közül kelljen választania; 2. a kezd?ponttól kívánt menüpontig való eljutáshoz szükséges gombnyomások várható értéke a lehet? legkisebb legyen. Ehhez az el?z? év adataiból meg is becsülték az egyes menüpontok népszer?ségét.

A feladat: a Jóltervez? Osztály munkatársaként a menüpont-népszer?ségek birtokában alakítson ki az irányelvnek megfelel? menürendszert. A bemenet minden sorában a menüpontok száma és az egyes menüpontok relatív népszer?sége olvasható, például

```
5 2 2 1 3 12
```

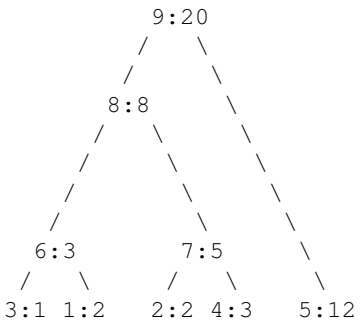
azt jelenti, hogy 5 menüpont van, melyekre annak valószn?sége, hogy egy telefonáló épp az adott menüpontot keresi, az 1. menüpont esetén 2/20, a 2. menüpont esetén 2/20, a 3. menüpont esetén 1/20, a 4. menüpont esetén 3/20 és az 5. menüpont esetén 12/20. Feltételezheti, hogy kevesebb, mint egymillió menüpont van.

Az optimális menürendszert úgy alakítsa ki, hogy vegye fel a menüpontokat különálló csúcsként, és mindig a két legkisebb népszer?ség? csúcsot kösse össze: hozzon létre egy új csúcsot, melynek bal oldali levele a legkisebb népszer?ség? csúcs, jobb oldali levele pedig a második legkisebb népszer?ség? csúcs. (Egyenl? népszer?ség esetén a kisebb sorszámú csúcsot vegye kisebbnek.)

Az egyes menüpontokat `sorszám:népszer?ség` párral jelölve a példabeli eloszláshoz az alábbi kezdeti gráf tartozik:

```
1:2  2:2  3:1  4:3  5:12
```

A legkisebb értékek összekötögetésével az alábbi bináris fát kapjuk:




A gráf leírja a menürendszert: az ügyfél a gyökérb?l (a `9:`-es csúsból) indul, mindig két lehet?ség közül választ, és a megfelel? részfán megy tovább. Ha levélbe ér, eljutott a kívánt menüpontig (`1:` ... `5:` valamelyike).

A megírandó program a kimenetre a gráf csúcsait írja `össznépszer?ség,szül?,balgyerek,jobbgyerek` formátumban, nullát írva, ha az adott csúcsnak nincs szül?je, bal illetve jobb gyereke. Minta kimenet:

```
2,6,0,0 2,7,0,0 1,6,0,0 3,7,0,0 12,9,0,0 3,8,3,1 5,8,2,4 8,9,6,7 20,0,8,5
```

A fenti kimenetben pl. a 7. helyen álló `5,8,2,4` azt jelenti, hogy a fa 7-es csúcsának össznépszer?sége 5, szül?je a 8-as csúcs, balgyereke a 2-es csúcs, jobbgyereke pedig a 4-es csúcs. (Ez az ábráról is leolvasható.)

Nem kell foglalkoznia annak bizonyításával, hogy a vázolt algoritmus valóban optimális eredményt szolgáltat. (Ha az ellen?rz?csoport kételyét fejezné ki, hivatkozzon a http://en.wikipedia.org/wiki/Huffman_coding oldalra.)

A megírandó program (~/szimp2/boldogugyfel.c) vázlata:

```
#include <stdio.h>

struct csucs {
  int sorszam, nepszeruseg;
  struct csucs *bal, *jobb, *szulo;
};

#define NAGYON_NAGY 2000000000

/** t[0] nemlétez? csúcs, sorszáma nulla, népszer?sége NAGYON_NAGY.
 * Azért kell kétmillió elem, mert majdnem egymillió levele és majdnem
 * egymillió bels? csúcsa lehet a fának.
 */
struct csucs t[2000000];

/** A valódi csúcsok száma (t[1] ... t[n]) */
int n;

/** A szül?vel nem rendelkez? csúcsok közül megkeresi a két legkisebb
 * össznépszer?ség? csúcsot. A legkisebbet bp-be, a második legkisebbet
 * jp-be teszi. Egyenl?ség esetén a kisebb sorszám számít.
 */
void ket_legkisebbet_keres(struct csucs **bp, struct csucs **jp) {
  *bp=*jp=&t[0];
  for (i...) {
    ... /* ha van szül?je, kihagyjuk */
    ... /* egyébként, ha bp népszer?ségénél kisebb, betesszük bp-be... */
    ... /* egyébként, ha jp népszer?ségénél kisebb, betesszük jp-be */
  }
}

int main(void) {
  struct csucs *b, *j;

  /* a t[0] strázsa inicializálása */
  t[0].sorszam=0;
  t[0].nepszeruseg=NAGYON_NAGY;
  t[0].bal=t[0].jobb=t[0].szulo=NULL;

  while (1==scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu,&n)) {

    for (i...) { /* bementi népszer?ségek olvasása */
      t[i].sorszam=i;
      scanf(...);
      ...
    }

    for (i...) { /* n-1 db összekötés */
      ket_legkisebbet_keres(...);
      n++;
      ...
    }

    for (i...) { /* a fa kiírása */
      printf("http://wiki.math.bme.hu%d,%d,%d,%d "http://wiki.math.bme.hu, ...);
    }
    printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
```

```
  }
  return 0;
}
```

# 7. el?adás (2007-03-30)

A mutatókat ismételtük.

# 8. gyakorlat (2007-04-03, 2007-04-06)

A keddi gyakorlaton anagrammageneráló programot írtunk. Nem túl gyorsat. A kulcsfüggvény (az a függvény, melyre a legnehezebb rájönni):

```
char t[...];
/** A szó a t tömb els? n elemében foglal helyet */
int n;
/** Ha a t tömb els? i db eleme már a megfelel? helyén van, akkor el?állítja
 * és kiírja a maradék elemek összes permutációját (a kiírás az els? i elemmel
 * együtt történik.
 */
int permutal(int i) {
  int j;
  if (i<n) {
    for (j=i;j<n;j++) {
      csere(&t[i],&t[j]);
      permutal(i+1);
      csere(&t[i],&t[j]);
    }
  } else {
    ... /* a t tömb els? n elemének kiírása */
  }
}
```

Ez az ábra került a táblára (a fényképezésért köszönet Sz?cs Benedeknek):

http://www.math.bme.hu/~pts/szimp2/permutal.jpg

# Ami a C-s el?adásokból kimaradt

Hogyan lehet kirajzolni egy bináris fát? A legegyszer?bb rekurzívan:

```
void kirajzol(struct csucs *gyoker, int nszokoz) {
  int i;
  for (i=nszokoz; n>0; n--) putchar(' ');
  if (gyoker!=NULL) {
    printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, (*gyoker).ertek);
    kirajzol((*gyoker).bal,  nszokoz+2);
    kirajzol((*gyoker).jobb, nszokoz+2);
  } else printf("http://wiki.math.bme.hu-\n"http://wiki.math.bme.hu);
}
```

A fenti megoldás könyvtárstruktúraszer?en rajzolja ki a fát, például az

```
    5
   / \
  4   6
 /     \
```

```
3       7
```

fát így rajzolja ki:

```
5
  4
    3
      -
      -
    -
  6
    -
    7
      -
      -
```

Az alábbi anyagrészt hasznos tudni, de nem kerül számonkérésre.

Hogyan lehet lemásolni egy bináris keres?fát? Ehhez szükség van annyi szabad memóriára, amennyit a másolat számára szükséges. Memóriafoglaláshoz a malloc függvényt használhatjuk. A `malloc(n)` hívás lefoglal egy n hosszúságú memóriaterületet, és visszadja a kezd?címét. Felszabadítható a free függvénnyel, vagy ha elmulasztjuk, akkor az operációs rendszer felszabadítja, amikor a program kilép.

T típusú memóriaterület foglalása és felszabadítása:

```
#include <stdlib.h> /* a malloc() miatt kell */
...
  T *p;
  ...
  p=(T*)malloc(sizeof(T));
  ...
  free(p);
```

Mostmár le tudjuk másolni a fát:

```
/** Lemásolja a részfát, és visszadja a másolat gyökerének címét. */
struct csucs *lemasol(struct csucs *gyoker) {
  /** Az mgyoker a másolat gyökere. */
  struct csucs *mgyoker;
  if (gyoker==NULL) return NULL;
  mgyoker=(struct csucs*)malloc(sizeof(struct csucs));
  (*mgyoker).ertek=(*gyoker).ertek;
  (*mgyoker).bal =lemasol((*gyoker).bal );
  (*mgyoker).jobb=lemasol((*gyoker).jobb);
}
```

# 8. el?adás és további el?adások

Ez már objektumorientált programozásról és Rubyról szól, és a SZIMP2 Ruby oldalon olvasható.

# 9. gyakorlat és további gyakorlatok

Ez már objektumorientált programozásról és Rubyról szól, és a SZIMP2 Ruby oldalon olvasható.

# Az els? ZH

## Id?pontok (pótZH is)

A SZIMP2 els? ZH id?pontja: 2007. április 13. péntek, 12:00, helye a K.mf.65. 90 perces lesz, bruttó 120 perc. Aki számítógépen oldja meg a programozási feladatot, annak az el?z?ek mellett további 90 perc (14:00--16:00, H.57). Konzultáció: 2007. április 10. kedd, 16:30--19:00, H.27. PótZH 7 nappal kés?bb, a K.3.14-ben, ugyanilyen feltételek szerint.

Már nem igaz, hogy ZH-ra csak az jöhet, aki a ZH el?tti napon (vagy korábban) az összes kötelez?, C nyelv? HF-et OK-san beadta a SIO-ban. Az viszont igaz, hogy aki nem adta be, az jöhet ZH-t írni, a HF-et pedig a szorgalmi id?szak végéig pótolhatja.

Az e-mailes jelentkezés alapján a ZH programozási részét számítógépen írhatják (a jelentkezés sorrendjében):

- Er? Zsolt
- Szabó Viktor
- Koszta Botond
- Cziráki Tamás
- Barta Zsuzsanna
- Gubek Andrea
- Winkler László
- Vajda István
- Urbán Eszter
- Sepsi Róbert

A ZH 90 perces lesz, papíralapú. 80% elmélet, 20% programozás. A téma: programozási alapfogalmak és azok alkalmazása C nyelven. Egyetlen programozási feladat lesz C nyelven, egy tízsoros függvényt kell majd megírni. Az elméleti részben is feladatmegoldás lesz. A ZH-n segédeszközként használható a *Standard C reference* + 2 db kézzel, tollal írt A4-es lap. A lapot is be kell adni a ZH-val (de a hallgató a ZH javítása után visszakapja a lapot).

Ha valaki a programozási feladathoz szeretne segédeszközként számítógépet használni, az jelezze e-mailben Szabó Péternek legalább 100 órával a ZH kezdete el?tt. Ebben az esetben számára a ZH jóval hosszabb lesz (az eredeti 90 perc helyett akár 90+90 percig is elhúzódhat, a felmerül? technikai nehézségek függvényében).

## Eredmények

A ZH-n az alábbi 33 db pontszám született: 23 40 44 44 54 62 63 64 64 67 70 71 71 72 73 75 75 79 83 86 87 89 89 92 93 93 95 100 102 102 103 109 110.

Gratulálok a szép eredményekhez!

A @math.bme.hu-s e-mail-címére mindenkinek elküldtem a saját pontszámát. A dolgozatok megtekinthet?k és a puskák visszavehet?k a keddi gyakorlat el?tt (14:00, H.27). A jegyek tájékoztató jelleg?ek, a félév végén a ZH-k összpontszáma számít majd. Jegytáblázat:

- p<50: elégtelen
- 50<=p && p<62: elégséges
- 62<=p && p<75: közepes

- 75<=p && p<89: jó
- 89<=p: jeles

Azoknak kell pótZH-t írniuk, akinek nincs érvényes, legalább 50 pontos ZH-eredményük. A pótZH a ZH után pontosan 7 nap múlva lesz (a napon belül azonos id?beosztással), a terem foglalása folyamatban van. Ha valaki szeretné a ZH-eredményét töröltetni, és helyette pótZH-t írni, az írjon nekem e-mailt err?l.

# Az elméleti rész

A pontok 80%-át ebb?l a részb?l lehet szerezni.

Kizárólag alábbiakkal azonos típusú, az alábbiaknál nem nehezebb kérdésekre lehet számítani:

- Tekintsük a `struct nagy { int i; char c1, *c2; short s; char c3; double d; }` struktúratípust.
  - Mekkora a `struct nagy a[10][20][30], *b;` változók mérete igazítás (alignment) nélkül? Segítség: a char 1 bájt, a short 2 bájt, a double 8 bájt, a többi 4 bájt.
  - Mekkora a `struct nagy a[10][20][30], *b;` változók mérete igazítással? Segítség: a char igazítása 1 bájt, a short igazítása 2 bájt, a többi igazítása 4 bájt.
  - Csökkentse minimálisra a struktúra igazításos méretét pusztán a mez?k áthelyezésével. Mennyi az új méret?
- Milyen (geometriai) transzformációt végez a derékszög? koordinátarendszerbeli (x,y) ponton az `x-=y+=x-=y;` utasítás?
- Tekintsük a törtszámokat összeadó *osszead* függvényt:

```
... ...(...) {
  egyszerusit(as,an);
  egyszerusit(bs,bn);
  en=lkkt(an,bn);
  es=(en/an)*as+(en/bn)*bs;
  egyszerusit(es,en);
}
```

- - Egészítse ki a ...-os részeket. A paraméterek sorrendje tetsz?leges.
  - Pótolja a kihagyott `&` és `*` jeleket.
  - Változtat-e valamit az eredményen, ha az utolsó utasítást (`egyszerusit(es,en);`) kihagyjuk? A válasz csak indoklással együtt fogadható el.
- Tekintsük az alábbi függvényt egy am?bajátékot játszó programból. Az aktuális játékállás a *t* változóban található. A tábla 19x19 mez?b?l áll.

```
int feladvany1(int n, int m) {
  int x, y;
  char c;
  for (y=0;y<n;y++) {
    for (x=4;x<m;x++) {
      if (0==(c=t[y][x])) {
      } else if (c==t[y][x-1] && c==t[y][x-2] && c==t[y][x-3] && c==t[y][x-4]) {
        return c;
      }
    }
  }
  return 0;
}
```

- - Deklarálja a *t* változót.
  - Mit számol ki (pontosabban: mit ad vissza) a függvény?

- ♦ Mit tartalmaz a *t* változó a játék kezdetekor?
- ♦ Miért lenne rossz, ha a kódban `x=4` helyett `x=0` szerepelne?

- Tekintsük az alábbi függvényt. Minden kett?spontra végz?d? sorhoz adja meg, hogy mit tartalmaz a *t* tömb.

```
void feladvany2(void) {
  int t[9], *p, i;
  t[0]=0; t[1]=1;
 januar:
  for (i=2;i<9;i++) t[i]=t[i-1]+t[i-2];
 februar:
  p=&t[3];
 marcius:
  t[0]=*p++;
 aprilis:
  t[1]=*++p;
 majus:
  t[2]=(*p)++;
 junius:
  t[3]=++(*p);
 julius:
  t[4]=++*p;
 augusztus:
  *p+=*p;
 szeptember:
  ;
}
```

- Tekintsük az alábbi ciklust:

```
for(i=0;i<=100&&t[i]!=42;i++){}
```

- ♦ Deklarálja a változókat úgy, hogy a lehet? legkevesebb memóriát használja.
- ♦ Írja át a ciklust *for*-b?l *while*-ba.
- ♦ Valósítsa meg a ciklust *goto*-val.
- ♦ Rajzolja le a ciklus m?ködését folyamatábrán.
- ♦ A ciklus befejeztével mi az *i* változó értékének jelentése?
- ♦ A ciklus befejeztével az *i* változó milyen esetben veszi fel a maximumát?

- Tekintsük az alábbi programrészletet:

```
void helyben_transzponal(int t[9][9])
int x, y
double d, *p
for (y=0;y<9;y++)
p=&t[y][0]
for (x=y+1;x<9;x++)
d=p[x]
p[x]=t[x][y]
p[x]=d
```

- ♦ Pótolja a hiányzó pontosvessz?ket és kapcsos zárójeleket.
- ♦ Pótolja a tanult módon a hiányzó soreleji szóközöket (indentation).
- ♦ Lassít vagy gyorsít a függvény futásán, ha `y+1` helyett `y`-t írunk?
- ♦ Változtat-e a függvény futásának eredményén, ha `y+1` helyett `y`-t írunk?

- Tekintsük az alábbi függvényt:

```
int fib(int n) {
```

```
  if (n<1) return 0;
  return fib(n-1)+fib(n-2);
}
```

- ♦ A függvény az *n*-edik Fibonacci-számot adná vissza, ha helyesen m?ködne. Mi a hiba?
  ♦ Javítsa ki a hibát 2 karakter megváltoztatásával.
  ♦ A Fibonacci-sorozatot értelmezhetjük negatív *n*-ekre is (a rekurzív képlet ugyanaz, mint pozitívakra). Módosítsa a programot, hogy negatív *n*-ekre is m?ködjön. Háromsoros megoldást adjon.
- Tekintsük az alábbi *keres* függvényt, amely az *n* elemb?l álló, nagyság szerint rendezett *t* tömbben keresi meg az *e* értéket, és tekintsük az ?t hívó *feladvany3* függvényt.

```
int keres(int t[], int n, int e) {
  int a=0, b=n-1, f;
  while (a<=b) {
    f=(a+b)/2;
    if (e==t[f]) return f;
    else if (e<t[f]) b=f-1;
    else a=f+1;
  }
  return n;
}

int feladvany3(int t[], int n, int e) {
  int a=keres(t, n, e), b=a;
  while (a>0 && t[a-1]==e) a--;
  while (b<n && t[b]==e) b++;
  return b-a;
}
```

- ♦ M?ködik-e a *keres* függvény minden monoton csökken? tömbre? A válasz csak bizonyítással együtt fogadható el.
  ♦ M?ködik-e a *keres* függvény minden monoton növekv? tömbre? A válasz csak bizonyítással együtt fogadható el.
  ♦ Mit ad vissza a *keres* függvény, ha a keresett érték szerepel a tömbben?
  ♦ Megtalálja-e a *keres* függvény a keresett értéket, ha az többször is szerepel? Ha igen, melyik el?fordulását találja meg?
  ♦ Mit ad vissza a *keres* függvény, ha a keresett érték nem szerepel a tömbben?
  ♦ Mit számol ki (pontosabban: mit ad vissza) a *feladvany3* függvény, ha a keresett érték szerepel a tömbben?
  ♦ Mit ad vissza a *feladvany3* függvény, ha a keresett érték nem szerepel a tömbben?
- Tekintsük az alábbi deklarációt és függvényt.

```
struct csucs {
  int ertek;
  struct csucs bal , jobb;
};

int feladvany4 struct csucs gyoker {
  int bal , jobb;
  if gyoker == NULL return 0;
  return feladvany4 gyoker . bal + feladvany4 gyoker . jobb + gyoker . ertek;
}
```

- ♦ Pótolja a hiányzó `*`-okat és zárójeleket (a lehet? legkevesebbet).
  ♦ Mit számol ki (pontosabban: mit ad vissza) a függvény?
  ♦ Hányszor hívja meg önmagát a függvény egy *n* csúcsból álló fa esetén? Teljesen pontos választ adjon.

- Tekintsük az alábbi függvényt, ami 3 egész szám összesített eltérését határozza meg a közép? elemt?l:

```
int medianelteres(int a, int b, int c) {
  a=abs(a-kozepso(a, b, c));
  b=abs(b-kozepso(a, b, c));
  c=abs(c-kozepso(a, b, c));
  return a+b+c;
}
```

- ♦ A függvényben van egy alapvet? hiba, például a `medianelteres(7, 6, 5)` hívás eredménye 6, noha azt szeretnénk, hogy 2 legyen. Javítsa a hibát új változó bevezetésével.
  ♦ Javítsa a hibát új változó bevezetése nélkül. (Át kell írni az egész kapcsos zárójelen belüli részt esetszétválasztásosra.)
- Tekintsük az alábbi függvényt:

```
int feladvany5 int n {
int c = 0 int d = 2
if n < 0 n *= - 1
while n > 1{
if n % d == 0 {
n /= d c ++
} else {
++ d
}
}
return c
}
```

- ♦ Pótolja a hiányzó zárójeleket és pontosvessz?ket.
  ♦ Pótolja a tanult módon a soreleji szóközöket (indentation).
  ♦ Mit számol ki (pontosabban: mit ad vissza) a függvény?
- Deklaráljon egy függvényt, ami kiszámolja három egész szám legnagyobb közös osztóját.
- Deklaráljon egy függvényt, ami b?vít egy törtet (számláló,nevez?) egy adott számmal.
- Deklaráljon egy függvényt, ami összehasonlít két bináris keres?fát, hogy egymás tükörképei-e. El?bb definiálja a használt struktúratípust.
- Deklaráljon egy függvényt, ami lemásol egy bináris keres?fát, és visszaadja a másolatot. El?bb definiálja a használt struktúratípust.
- Deklaráljon egy változót, ami egy int-ekb?l álló háromdimenziós tömb.
- Deklaráljon egy változót, ami egy short-okra mutató mutatókból álló kétdimenziós tömb.

Az alábbiakhoz hasonló, tisztán elméleti feladatok nem lesznek:

- Mi a különbség a predekrementálás és a posztdekrementálás között?
- Mik a C nyelv kulcsszavai?
- Hogyan kell paraméterezni a scanf függvényt?
- Mi a strúktúratípus-definíció formális szintaxisa?
- Hogyan lehet egy függvényb?l két értéket visszaadni?

# A programozási feladat

A programozási feladat helyes megoldásáért a ZH-pontszámok 20%-a szerezhet?.

Egyetlen programozási feladat lesz, egy kb. 10 soros függvényt kell majd írni, ami hasonló lesz az el?adáson, a gyakorlaton vett és a házi feladatokban kit?zött fügvényhez, ám azoknál jóval egyszer?bb lesz. Példák:

- Írjon olyan függvényt C nyelven, ami visszadja egy bináris fában található csúcsok összegét. Deklaráció: `int ertekeket_osszead(struct csucs *gyoker);`
- Írjon olyan függvényt C nyelven, ami összehasonlít két bináris fát, és 0-t ad vissza, ha megegyeznek, és 1-et, ha különböznek. Az egyezéshez nem elég, hogy ugyanazokat az értékeket tartalmazzák, a gráfnak is meg kell egyeznie. Deklaráció: `int osszehasonlit(struct csucs *gyoker1, struct csucs *gyoker2);`
- Írjon olyan függvényt C nyelven, ami visszadja egy 1-nél nagyobb egész szám legkisebb, 1-nél nagyobb (prím)osztóját. Deklaráció: `int minoszto(int n);`. Az osztó keresését 2-t?l négyzetgyök n-ig végezze.
- Írjon olyan függvényt C nyelven, ami visszadja az n karakterb?l álló t tömbben a leghosszabb szó hosszát (ami 0 is lehet, ha egy szó sincs a tömbben). Szónak számít egy szóközt nem tartalmazó, tovább nem b?víthet?, összefügg? részsorozat. Deklaráció: `int maxszohossz(char t[], int n);`
- Írjon egy olyan rekurzív függvényt C nyelven, ami kiszámolja az n alatt a k binomiális együtthatót modulo m. Deklaracio: `int binom(int n, int k, int m);` Használja ki, hogy 0 alatt a 0 egyenl? 1-gyel, egyébként 0 alatt az n egyenl? 0-val, egyébként pedig n alatt a k egyenl? (n-1) alatt a k plusz (n-1) alatt a (k-1).
- Írjon olyan függvényt C nyelven, ami keres a 100-szor 100-as m mátrixban két azonos sort, és visszadja azok sorszámát (0 <= i < j < 100). Ha nincs két azonos sor, akkor i=j=100-at ad vissza. Deklaráció: void azonos_sort_keres(int m[100][100], int *i, int *j);</code>
- Írjon olyan függvényt C nyelven, ami kiszámolja három szám legnagyobb közös osztóját. Írhat segédfüggvényt is. Deklaráció: `int lnko3(int a, int b, int c);`
- Írjon olyan függvényt C nyelven, ami meghatározza, hogy egy háromszög derékszög?-e, és 0-et ad vissza, ha nem, 1-et, ha az A csúcsnál van derékszög, 2-t, ha a B csúcsnál, és 3-at, ha a C csúcsnál. A háromszög 3 csúcspontjával van megadva, minden koordináta egész szám. A derékszög?séget vektorok skalárszorzatával határozza meg. Deklaráció: `int derekszogu_e(int ax, int ay, int bx, int by, int cx, int cy);`.

# Házi feladatok

A házi feladatokat a SIO rendszer segítségével kell beadni. Csak azokat kell beadni, amelyeknek a beadási határideje itt fel van tüntetve.

A többi SIO-beli feladatot szorgalmi feladatként lehet beadni. A szorgalmi feladatok megoldása elmélyíti a hallgató tudását, növeli a hallgató gyakorlati rátermettségét, és nagyban segíti a ZH-ra való felkészülést.

## Technikai részletek

A C fordításhoz használt szoftver a *gcc* (a pontos verzió: `gcc 4.1.2 Debian 4.1.1-21`). A félév elején még egy korábbi *gcc*-vel fordítottunk (`gcc 3.4.6 Debian 3.4.6-4)`). dd A házi feladatokat a következ? módon fordítjuk:

```
$ gcc -x c -O2 -std=c99 -pedantic-errors -static -o <prog> <forrásfájl>.c -lm
```

A feladat megoldásakor el?bb nem a fenti paranccsal, hanem az alábbival érdemes el?bb kipróbálni a fordítást:

```
$ gcc -W -Wall -s -O2 -std=c99 -pedantic-errors -lm -o <program> <program>.c
```

Ez azért jó, mert a `-W -Wall` sok hasznos figyelmeztet? üzenetet megjelenít, és ily módon a hibákra hamar felhívja a figyelmet.

# Kiírt házi feladatok

| kód | leírás | beadási határid? |
|---|---|---|
| uc | https://sioweb.math.bme.hu/user.phtml?op=inc&id=6 | 2007. március 31., 24:00 CET |
| lcuc | https://sioweb.math.bme.hu/user.phtml?op=inc&id=7 | 2007. március 31., 24:00 CET |
| pow2count | https://sioweb.math.bme.hu/user.phtml?op=inc&id=38 | 2007. március 31., 24:00 CET |
| primecount | https://sioweb.math.bme.hu/user.phtml?op=inc&id=205 | 2007. március 31., 24:00 CET |
| matmul | https://sioweb.math.bme.hu/user.phtml?op=inc&id=206 | 2007. március 31., 24:00 CET |
| binfabe | https://sioweb.math.bme.hu/user.phtml?op=inc&id=751 | 2007. április 12., 24:00 CET |
| unolight | https://sioweb.math.bme.hu/user.phtml?op=inc&id=752 | 2007. március 31., 24:00 CET |
| binfabeki | https://sioweb.math.bme.hu/user.phtml?op=inc&id=753 | 2007. április 12., 24:00 CET |
| tortszoroz | https://sioweb.math.bme.hu/user.phtml?op=inc&id=759 | 2007. április 12., 24:00 CET |
| sikidom | https://sioweb.math.bme.hu/user.phtml?op=inc&id=762 | 2007. május 10., 24:00 CET |
| binfaber | https://sioweb.math.bme.hu/user.phtml?op=inc&id=763 | 2007. május 10., 24:00 CET |

A házi feladatok listája elérhet? a SIO rendszerben is: https://sioweb.math.bme.hu/user.phtml?op=zadania

# Típushibák a házikban

- A *Compile error* oka lehet az is, hogy a beadott fájl kiterjesztése nem `.c` . Tipikus rossz kiterjesztések: `.cc`, `.cpp`, `.cxx`.
- Ha *runtime error*-nak lehet oka, hogy a `return 0;` hiányzik a *main* függvény végér?l.
- A forrásfájl utolsó sorának végén nincs soremelés (emiatt *compile error*).
- Nagy, 8 MB-nál nagyobb tömb van a main függvényen belül (emiatt *runtime error* és *segmentation fault*). Megoldás: a tömb deklarációját a függvényen kívülre tenni. Például így jobb:

```
char nagytomb[10000000];
int main(void) {
  ...
  return 0;
}
```

- A getchar() eredménye *char* típusú változóba kerül. Ez azért nem jó, mert a char csak 256 különböz? értéket tud tárolni, de nekünk 257 kéne (a 256 karakter, és a fájl végét jelz? -1). Megoldás: *int* típusú változóba tenni.
- Függvényen belül függvény van definiálva. Ez nem jó, a függvényeket csak egymás után szabad definiálni.
- A program fölösleges szóközöket ír ki (pl. a sor végére), és emiatt *wrong answer*-ös a megoldás. Ha azt kell kiírni, hogy hello, majd egy soremelést, akkor jó megoldás a `printf("http://wiki.math.bme.huhello\n"http://wiki.math.bme.hu);`, rossz megoldás a `printf("http://wiki.math.bme.huhello \n"http://wiki.math.bme.hu);`
- Az eratoszthenészi szitát while cikluson belül az összes feladványra végigszámolja, pedig elég lenne összesen egyszer. Emiatt *time limit exceeded*-es a megoldás.
- Tetsz?leges karaktert nem lehet beolvasni gets()-sel vagy fgets()-sel, mert elnyelik a 0-s kódú karaktereket. A helyes megoldás karakter beolvasásásra getchar().
- A program beszélgetni próbál a felhasználóval, pl. kiírja, hogy *Most kérem a mátrix magasságát*. Épp amiatt rossz (*wrong answer*) a kimenet, hogy ez az üzenet is szerepel benne.
- A program egymás alá próbálja igazítani számokat. Épp amiatt rossz (*wrong answer*) a kimenet, hogy az igazítás miatt fölösleges szóközöket ír ki.
- A program túl sok memóriát használ, például a szitáláshoz használt tömb elemeit *int*-ekb?l építi,

noha *char*-okba is beleférne. i386 architektúrán az int 32 bites (ebb?l 1 el?jel), a char 8 bites (ebb?l 1 el?jel), tehát egy charban a [-128,127] zárt intervallumból lehet szám.

- Túl nagynak veszi a tömböt (és amiatt *memory limit exceeded*-et kap). A feladatkiírásban pontosan meg van adva a bemeneti adatok értéktartománya. Pont akkora (vagy egy hajszálnyival nagyobb) tömböt kell deklarálni, amekkorába az adat belefér.
- A tömb mérete nem konstans. Például hibás az `int tomb[n];` deklaráció, és helyes az `int tomb[500];` deklaráció, ha *n* értéke legfeljebb 500 lehet (az értéktartományt lásd a feladat szövegében).