Tartalomjegyzék

1. gyakorlat (2008-02-12 és 2007-02-15)

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.

Linuxos környezet

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
```

Az els? C program

Az info2 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.

~/info2/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 (az ún. *behúzás* angolul *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 ~/info2
$ cd ~/info2
$ kate hello.c
$ gcc -W -Wall -s -O2 -o hello hello.c
$ ./hello
Hello, World!
```

A kate parancsot indíthatjuk menüb?l is, vagy külön terminálablakból, és akkor a következ? parancs kiadásához nem kell kilépni bel?le.

A gcc-parancssor elemeinek jelentése

Ezt nem kell tudni, csak tájékoztatásul került 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.

Windows alatt

DevC++ konfigurálása

Nyelv kiválasztása

- El?ször nyugodtan válasszuk a magyart mindkétszer, kés?bb azonban álljunk át angolra

- a dev-C++ gyorsbillenty?-beállításai ütik egymást a magyar billenty?zeten a pontosvessz? (AltGr+,) kombinációjával, ezért els? futtatáskor tedd a következ?t:

```
"http://wiki.math.bme.huEszközök"http://wiki.math.bme.hu -> "http://wiki.math.bme.huGyorsbille
"http://wiki.math.bme.huSzerkeszés:Megjegyzés Be"http://wiki.math.bme.hu ill. "http://wiki.mat
"http://wiki.math.bme.huESC"http://wiki.math.bme.hu<- (törli a gyorsbillenty?t)
"http://wiki.math.bme.huOk"http://wiki.math.bme.hu<-
```

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).

Teszt

Futtassuk le az alábbi programot mindkét oprendszer alatt!

~/info2/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 behúzás (beljebbezés) szabályai

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. Kezdetben kapcsoljuk ki, mint hogy figyelnünk kelljen a hibázásaira, és folyamatosan küzdenünk kelljen vele. Kés?bb, kipróbálhatjuk a következ? beállításokat:

Angol esetén

```
Auto Indent: checked
Use Tab Character: not checked
Smart Tabs: not checked
Keep Trailing Spaces: checked
Backspace Unindents: checked
Enhanced Home Key: checked
```

Magyar esetén

```
Automatikus bekezdés: pipa
Tab használata: üres
Rövid tabok: üres
Sorvégi "http://wiki.math.bme.huspace"http://wiki.math.bme.hu-ek megtartása: pipa
Visszatörléssel bekezdés ki: pipa
Módosított Home billenty?: pipa

Zárójelpárok kiemelése: pipa  (javasolt)

Tab mérete: 2
```

További példák

Hasznos parancsok:

```
$ cd ~/info2
$ 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
```

~/info2/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;
}
```

~/info2/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 oly módon, hogy `a' bináris alakját `b' bittel balra tolja, és visszaadja az eredményt. Hasonló 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 (az el?jelbitre csordulás) miatt vannak.

1. el?adás (2008-02-15)

A C nyelv

- története
  - C programozási nyelv: Dennis Ritchie 1972, Bell Telephone Laboratories
  - K&R C 1978 (Brian Kernighan, Dennis Ritchie: The C Programming Language)
  - ANSI (American National Standards Institute) 1989 -> C89
  - ISO (International Organization for Standardization) ISO/IEC 9899:1990 -> C90 (lényegében azonos)
  - ISO 9899:1999 (ANSI 2000) -> C99
  - hatás, utódok: Objective-C, C++, C#
- tulajdonságai
  - általános célú, blokkstruktúrált, imperatív (utasításokból áll, melyek megváltoztatják a program állapotát), procedurális (az el?bbi megvalósítása eljáráshívásokkal) nyelv
  - Unixra készült, ma szinte minden platformon
  - rendszerprogramok, beágyazott rendszerek, alkalmazási programok
  - alacsony szint? memóriahozzáférés, hatékonyan fordul gépi kódra
  - támogatja a gépfüggetlen programozást

Fordítás

A fordítás lépései

```
gcc -E hello.c >hello.ee    (csak az el?feldolgozó fut le  -> standard outputra ír)
gcc -S hello.c              (fordít, de az assembler nem   -> hello.s -- assembly kód)
gcc -c hello.c              (fordít, de nem szerkeszt      -> hello.o -- object file)
gcc hello.c                 (el?feldolgoz+fordít+szerkeszt -> a.out -- futtatható állomány)
```

A -o opció nélkül a.out lesz a futtatható állomány neve.

Példák

Állománymásoló három változatban:

~/info2/allomanymasolo1.c

```
#include <stdio.h>
int main (void)
{
  int c;
  c = getchar();
  while (c != EOF) {
    putchar(c);
    c = getchar();
  }
  return 0;
```

```
}
```

~/info2/allomanymasolo2.c

```
#include <stdio.h>
int main (void)
{
  int c;
  while ((c = getchar()) != EOF) putchar(c);
  return 0;
}
```

~/info2/allomanymasolo3.c

```
#include <stdio.h>
main ()
{
  int c;
  while ((c = getchar()) != EOF) putchar(c);
}
```

Bájtszámláló

~/info2/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
```

Sorszámláló

~/info2/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
```

~/info2/szoszamlalo.c (ld. K&R alapján)

```
/* Megszámolja és kiírja, hogy a bemenet hány bájtból,
 * hány szóból és hány sorból áll. Szó a bemenet olyan
 * maximális része, melyben nincs szóköz, tabulátor és
 * újsor karakter.
 */
#include <stdio.h>
#define KINN 0
#define BENN 1
int main(void) {
  int c, bajtok_sz, szavak_sz, sorok_sz, allapot=KINN;
  bajtok_sz=szavak_sz=sorok_sz=0;
  while ((c=getchar()) != EOF) {
    ++bajtok_sz;
    if (c == '\n') sorok_sz++;
    if (c == ' ' || c == '\n' || c == '\t')
      allapot = KINN;
    else if (allapot == KINN) {
      allapot = BENN;
      szavak_sz++;
    }
  }
  printf("http://wiki.math.bme.hu%d %d %d\n"http://wiki.math.bme.hu, bajtok_sz, szavak_sz, sorok_s
  return 0;
}
```

2. gyakorlat (2008-02-19, 22)

SIO megismerése

https://sioweb.math.bme.hu

Tasks Submit solution My submissions

Change password Logout

Available contests

Bemenet fájlból teszteléshez

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)

Feladat (ember gondol, gép talál ki): Írj C programot, mely egy 'a' és 'b' (legyen pl. a=1, b=100) közé es? számot kitalál az általa föltett kérdésekra adott igen/nem (i/n) válaszokból!

~/info2/embergondol.c

```
#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.

Szám beolvasása

~/info2/duplazas.c

```
#include <stdio.h>

int main(void){
  long int tmp;
  while (1==scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu,&tmp)){
    tmp=tmp*2;
    printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu,tmp);
  }
  return 0;
}
```

2. el?adás (2008-02-22)

Változók, alap adattípusok, konstansok

Utasítások

Vigyázzunk a pontosvessz?kre

```
if (a > b);
  b = a;

if (a > b)
  b = a;

do {
  ...
} while (a < b);

while (a < b) {
  ...
}
```

M?veletek, relációk

++ --
: csak változó el?tt/után

i---j
: mohó lexikális elemzés miatt = i-- - j

---i
: értelmetlen (mohó algoritmus miatt = -- -i)

bitm?veletek
: ~ bitenkénti NOT
: & | ^ AND, OR, XOR

logikai m?veletek
: && || !

Bitszámlálás*

Számoljuk meg az unsigned i, egészként deklarált szám 2-es számrendszerbeli alakjában az 1-es biteket.

```
unsigned i;
int sz;
...
for (sz=0; i != 0; i >>= 1) if (i & 01) sz++;
...
```

Ha i 2-es komplemens alakban van tárolva, akkor i&(i-1) az utolsó 1-es bitet törli (MIÉRT?). Ezt kihasználva cseréljük ki a fenti sort egy vele ekvivalens, de gyorsabbra (amely ráadásul int-nek deklarált i-re is jól fut).

Precedencia (m?veletek végrehajtási sorrendje)

```
if ( a&04 ) ... ;

if ( a&04 != 0 ) ... ;
if ( a & (04!=0) ) ... ;

bajt = felso<<4 + also;
bajt = felso << (4 + also);
```

```
bajt = (felso<<4) + also;
bajt = felso<<4 | also;

if ( ( (ev%4 == 0) && (ev%100 != 0) ) || (ev%400 == 0) ) printf("http://wiki.math.bme.huszokoev"ht
if ( (ev%4 == 0 && ev%100 != 0) || ev%400 == 0 ) printf("http://wiki.math.bme.huszokoev"http://wik
if ( ev%4 == 0 && ev%100 != 0 || ev%400 == 0 ) printf("http://wiki.math.bme.huszokoev"http://wiki.
```

Kiértékelési sorrend

Egy <= irány

```
a = b = c = 0;
```

Egy => irány

```
if ( n!=0 && osszeg/n < atlag ) ... ;
if ( n==0 || osszeg/n < epszilon ) ... ;
```

Lógó else

Helyes:

```
if ( a == 0 )
  if ( b == 0 )
    return 0;
  else {
    c = f(a,b);
    return c;
  }
```

Helytelen:

```
if ( a == 0 )
  if ( b == 0 )
    return 0;
else {
  c = f(a,b);
  return c;
}
```

Az else mindig a legközelebbi else-nélküli if-hez tartozik!

Javítás: (a kapcsos zárójelek használata segít)

```
if ( a == 0 ) {
  if ( b == 0 )
    return 0;
} else {
  c = f(a,b);
  return c;
}
```

Tömbök

Asszimetrikus korlátok

Írjunk Mapleben 1-t?l, majd k-tól 7-elem? ciklust:

```
for i from 1 to 7 do ... end do;
for i from k to k+7-1 do ... end do;
```

A tömbök indexelése 1-t?l! Szimmetrikus intervallum! Az intervallum hossza = fels? határ - alsó határ + 1.

```
- + - - - - - - + - - -
 0 1 2 3 4 5 6 7 8 9
   ^           ^
- + - - - - - - + - - -
 1 2 3 4 5 6 7 8 9
   ^           ^
```

Írjunk C-ben 0-tól, majd k-tól 7-elem? ciklust! Az intervallum hossza = fels? határ - alsó határ.

```
for ( i=0; i<7; i++) ... ;
for ( i=k; i<k+7; i++) ... ;
```

A tömbök indexelése 0-tól! Aszimmetrikus intervallum!

```
- + - - - - - - + - -
   0 1 2 3 4 5 6 7 8
   ^             ^
- + - - - - - - + - -
 1 2 3 4 5 6 7 8 9
   ^             ^
```

Tömbök definiálása, 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
- 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
  - `#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

~/info2/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
...
61
$ sort -n | diff rendezett -    # és beírjuk pontosan ugyanazokat a számokat
86
36
16
...
61
# és itt semmit sem szabad kiírnia, hiszen nincs eltérés a Unixos és az
# általunk írt rendezés eredménye között
```

~/info2/matrixtranszponalas.c:

```
#include <stdio.h>
#define SOR 3
#define OSZ 4

int main(void) {
  int a[SOR][OSZ], b[OSZ][SOR], n, m;

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

  for (n=0; n < SOR; n++)
    for(m=0; m < OSZ; m++)
      b[m][n] = a[n][m];

  for (n=0; n < OSZ; n++) {
    for(m=0; m < SOR; m++) {
      printf("http://wiki.math.bme.hu%6d "http://wiki.math.bme.hu, b[n][m]);
      /* persze kiirhatnank a[m][n]-et is, es akkor nem is kell transzponalni */
    }
    printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
  }
  return 0;
}
```

Segédtömb használata nélkül transzponálja a mátrixot:

~/info2/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 (2008-02-26, 29)

1. Feladat: Írjuk ki az egymilliónál kisebb prímszámokat az Eratoszthenészi szita segítségével ~/info22/erszita1.c néven.

~/info2/erszita1.c:

```
#include <stdio.h>
int main (void) {
  int i,j;
  char tomb[1000000];



  return 0;
}
```

Kipróbálás:

```
$ cd ~/info2
$ 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
```

Egy lehetséges megoldás:

```
#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;
}
```

2. Feladat: Ezután javítsunk 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 kihúzta;
- 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.

~/info2/erszita2.c:

```
#include <stdio.h>
char tomb[1000000];
int main (void) {
  int i,j;



  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.

Egy lehetséges megoldás:

```
#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;
}
```

Szorgalmi feladat: Az eratoszthenészi szitát csak a páratlan számokra alkalmazzuk. 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 írni, és kettesével ugrálni.

~/info2/erszita3.c

```
#include <stdio.h>
char tomb[500000];
int main (void) {
  int i,j;



  return 0;
}
```

Egy megoldás:

```
#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;
}
```

3. el?adás (2008-02-29)

Függvényekkel kapcsolatos alapfogalmak:

- függvény, visszatérési érték, paraméter
- függvény definíciója

```
visszatérési_érték_típusa  függvénynév ( formális paraméterek listája vessz?vel elválasztva )
{
  utasitasok;
}
```

- ♦ a blokkok egymásba ágyazhatók, de függvény nem definiálható függvényben
- ♦ minden itt definiált változó *lokális változó*
- ♦ a visszatérési érték nem lehet tömb vagy függvény, de lehet ezeket megcímzõ mutató,

- prototípus

```
visszatérési_érték_típusa  függvénynév ( formális paraméterek listája vessz?vel elválasztva );
```

- függvényhívás

```
függvénynév ( aktuális paraméterek listája vessz?vel elválasztva );
```

- ♦ ha a függvény definíciója az els? hívása mögött van, szokás a prototípusát legel?re írni (különben warning):

~/info2/negyzet.c

```
#include <stdio.h>
int negyzet(int x);
int main(void) {
  printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, negyzet(15));
  return 0;
}
int negyzet(int x) {
  return x*x;
}
```

de persze az is jó, hogy

```
#include <stdio.h>
int negyzet(int x) {
  return x*x;
}
int main(void) {
  printf("http://wiki.math.bme.hu%d\n"http://wiki.math.bme.hu, negyzet(15));
  return 0;
}
```

- ♦ *formális paraméter* ill. *aktuális paraméter* más elnevezéssel *paraméter* ill. *argumentum*

- érték szerinti paraméterátadás -- cím szerinti paraméterátadás
- rekurzió

Érték és cím szerinti paraméterátadás

~/info2/parameteratadas.c

```
#include <stdio.h>

void f (int x)       // x egész
{
  x++;
}

void g (int *px)     // px egy cím, ahol egy egész van tárolva
{                    // amelyre *px hivatkozik
  *px=*px+1;
}

int main (void)
{
  int x=3;
  f(x);              // f nem változtatja meg x értékét
  printf("http://wiki.math.bme.hux f utan = %d\n"http://wiki.math.bme.hu,x);
                     // x címét &x jelöli
  g(&x);             // g megváltoztatja x értékét
  printf("http://wiki.math.bme.hux g utan = %d\n"http://wiki.math.bme.hu,x);
  return 0;
}
```

Két változó értékének cseréje

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:

```
#include <stdio.h>

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;
}
```

~/info2/csere.c -- ez már jó:

```
#include <stdio.h>

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;
}
```

Változók

Lokális és globális változók

Változók hatóköre (scope)

~/info2/scope.c

```
#include <stdio.h>

int main(void)
{
  int i = 0;                              // i a küls? blokkban
  do {
    int i = 0;                            // egy másik i
    ++i;                                  // és ez erre vonatkozik
    printf("http://wiki.math.bme.hui = %d\n"http://wiki.math.bme.hu, i);
  } while( ++i < 5 );                     // ez megint a küls? i
  printf("http://wiki.math.bme.hui = %d\n"http://wiki.math.bme.hu, i);
  return 0;
}
```

Statikus változók

~/info2/static.c

```
/* statikus <-> automatikus változók */
#include <stdio.h>

void fv1(void);
void fv2(void);

int main(void)
{
  int i;
  for(i = 0; i < 4; i++ )
  {
    fv1();
    fv2();
  }
  return 0;
}

/* fv1 automatikus változóval */
void fv1(void)
{
  int szamlalo = 0;
  printf("http://wiki.math.bme.hufv1:          szamlalo = %d\n"http://wiki.math.bme.hu, ++szamlalo
}

/* fv2 statikus változóval */
void fv2(void)
```

```
{
  static int szamlalo = 0;
  printf("http://wiki.math.bme.hufv2: statikus szamlalo = %d\n"http://wiki.math.bme.hu, ++szamlalo
}
```

Rekurzió

Példa a faktoriális rekurzív számolására (~/info2/fakt.c):

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

Példa a faktoriális rekurzív számolására akkumulátorral és jobbrekurzióval (~/info2/fakt2.c; nem kell érteni, hogy miért jobb a fakt2, mint a fakt):

```
#include <stdio.h>

unsigned long fakt2b(int n, unsigned long szorzo) {
  if (n<2) return szorzo;
  return fakt2b(n-1, szorzo*n);
}

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

int main(void)
{
  int n;
  while(1 == scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu, &n)){
    printf("http://wiki.math.bme.hu%lu\n"http://wiki.math.bme.hu,fakt2(n));
  }
  return 0;
}
```

Példa a Fibonacci-sorozat rekurzív számolására

~/info2/fib.c

```
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

~/info2/fib2.c

```
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 hatványozás rekurzív számolására (~/info2/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 (~/info2/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 (~/info2/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;
```

4. gyakorlat (2008-03-04, 2008-03-07)

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

Feladat: 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 történ 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 két történ *as*/*an*, *bs*/*bn* formában (valójában négy egész számot), majd kiszámolja az *as*/*an' és* bs/bn *történ összegét, és perjellel elválasztva kiírja az eredmény számlálóját (*es*) és nevez?jét (*en*) egy sorba. Ezután a következ? feladványsor beolvasásásval folytatja.*

Példa bemenet (~/info2/tortad1.in):

```
1/2 3/4
3/4 5/-6
3/4 -5/6
-3/4 5/6
3/6 30/20
-15579/-25862 9268/-90517
```

Példa kimenet (~/info2/tortad1.exp):

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

A megírandó program vázlata (~/info2/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 b==0, akkor a, 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 (~/info2/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 (b==0) return a;
  return lnko(b, a%b);
}

/* 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 (~/info2/tortad1.in) és a mintakimenetet (~/info2/tortad1.exp), lásd fent. Ezután kipróbálható:

```
$ cd ~/info2
$ gcc -W -Wall -s -O2 -o tortad tortad.c
$ ./tortad <tortad1.in
5/4
-1/12
-1/12
1/12
2/1
1/2
$ ./tortad <tortad1.in >tortad1.out
$ diff tortad1.exp tortad1.out
```

4. el?adás (2008-03-07)

Karakterláncok (sztringek)

- karakterlánc = karakterek tömbje, melyet egy 0-kódú karakter zár. Pl.

```
E   z   \n   e   g   y       \"http://wiki.math.bme.hu   k   a   r   a   k   t   e   r   l   a   n
```

```
69 122 10 101 ...      32  34 ...                                                  0
```

- Karakterlánc deklarálása inicializással:

```
char str[] = "http://wiki.math.bme.hu6 betu"http://wiki.math.bme.hu;
```

vele ekvivalens a következ? (miért 7?):

```
char str[7] = "http://wiki.math.bme.hu6 betu"http://wiki.math.bme.hu;
```

- Karakterlánc scanf-fel való beolvasásánál nincs & a tömb neve el?tt (miért? ld. kés?bb).

Az alábbi programban összef?zünk két karaketrláncot úgy, hogy az els? végér?l levesszük a '\0' karaktert, majd odamásoljuk a második karakterláncot, amit '\0'-val zárunk.

~/info2/osszefuz.c

```
#include <stdio.h>

int main(void)
{
  char str1[60]="http://wiki.math.bme.huEz\negy \"http://wiki.math.bme.hukarakterlanc\"http://wiki
  //  char str1[60]="http://wiki.math.bme.huEz egy \"http://wiki.math.bme.hukarakterlanc\"http://w
  char str2[60]="http://wiki.math.bme.hu, ez pedig egy masik, amiben van \\."http://wiki.math.bme.
  unsigned sz1 = 0;
  unsigned sz2 = 0;

  while (str1[sz1]) sz1++;   // az els? karakterlánc hossza
  while (str2[sz2]) sz2++;   // a második karakterlánc hossza

  if(sizeof str1 < sz1 + sz2 + 1)
    printf("http://wiki.math.bme.hu\nAz osszefuzes nem fog menni.\n"http://wiki.math.bme.hu);
  else {
    sz2 = 0;
    while(str2[sz2]) str1[sz1++] = str2[sz2++];

    str1[sz1] = '\0';        // 0-val zárjuk a karakterláncot
    printf("http://wiki.math.bme.hu\n%s\n"http://wiki.math.bme.hu, str1);
  }
  return 0;
}
```

Feladat: Olvassuk be valaki nevét és korát vessz?vel elválasztva, majd írjuk ki a nevét és a korát, de az utóbbiból tagadjunk le 10 évet! Használjuk a %[...] formátumot és a `stdlib.h` `atoi` nev? függvényét!

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

int main(void)
{
  char nev[60];
  char kor[4];
  int sz1 = 0, sz2 = 0, k = 0;

  scanf("http://wiki.math.bme.hu%[^,], %[0123456789]"http://wiki.math.bme.hu, nev, kor);
  while (nev[sz1]) sz1++;   // az els? karakterlánc hossza
  while (kor[sz2]) sz2++;   // a második karakterlánc hossza
  k=atoi(kor)-10;

  printf("http://wiki.math.bme.huNeved: %s, korod: %d\n"http://wiki.math.bme.hu, nev, k);
```

```
  return 0;
}
```

Mutató (pointer)

- Mutató deklarálása:

```
int *pn;
```

- NULL egy (több headerfájlban is definiált) konstans cím, mely a memóriában nem mutat sehová.

```
int *pn=NULL;
```

- A NULL cím egyszer?en tesztelhet?, vagyis hogy egy mutatónak adtunk-e már valódi címet: az `if(pn==NULL)` ekvivalens a `if(!pn)` kóddal.
- & a címet adó unér m?velet:

```
int n;
int *pn=&n;
```

- (*pn)++ nem ugyanaz, mint *pn++ a precedenciaszabályok miatt.
- pn++ a *pn típusának megfelel? bájtértékkel növeli a mutató értékét, tehát pl. char esetén 1-gyel, int esetén 4-gyel.
- a scanf függvény mutatót vár, ezért a következ?képp olvashatunk be egész számot:

```
int n = 0, *pn;
pn=&n;
scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu,pn);
```

vagy

```
scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu,&n);
```

Az alábbi példában az n változóra mutat a pn mutató. Vizsgáljuk meg az értékeiket a (*pn)++; és a pn++; utasítások után!

~/info2/mutato.c

```
#include <stdio.h>
int main(void)
{
  int n, *pn = NULL;  // pn egy mutató, mely egy egészre mutat
  n = 5;
  pn = &n;

  printf("http://wiki.math.bme.hu\nn címe:     %p\n"http://wiki.math.bme.hu, &n);
  printf("http://wiki.math.bme.hun mérete:   %d\n"http://wiki.math.bme.hu, sizeof n);
  printf("http://wiki.math.bme.hun értéke:   %d\n"http://wiki.math.bme.hu, n);
  printf("http://wiki.math.bme.hupn címe:    %p\n"http://wiki.math.bme.hu, &pn);
  printf("http://wiki.math.bme.hupn mérete:  %d\n"http://wiki.math.bme.hu, sizeof pn);
  printf("http://wiki.math.bme.hupn értéke:  %p\n"http://wiki.math.bme.hu, pn);
  printf("http://wiki.math.bme.hu*pn értéke: %d\n"http://wiki.math.bme.hu, *pn);

  (*pn)++;
  printf("http://wiki.math.bme.hu    (*pn)++;\n"http://wiki.math.bme.hu);
  printf("http://wiki.math.bme.hupn értéke:  %p\n"http://wiki.math.bme.hu, pn);
  printf("http://wiki.math.bme.hu*pn értéke: %d\n"http://wiki.math.bme.hu, *pn);
```

```
  pn++;
  printf("http://wiki.math.bme.hu    pn++;\n"http://wiki.math.bme.hu);
  printf("http://wiki.math.bme.hupn értéke:  %p\n"http://wiki.math.bme.hu, pn);
  printf("http://wiki.math.bme.hu*pn értéke: %d\n"http://wiki.math.bme.hu, *pn);

  return 0;
}
```

Egy futás eredménye:

```
n címe:     0xbfa8d2f0
n mérete:   4
n értéke:   5
pn címe:    0xbfa8d2ec
pn mérete:  4
pn értéke:  0xbfa8d2f0
*pn értéke: 5
    (*pn)++;
pn értéke:  0xbfa8d2f0
*pn értéke: 6
    pn++;
pn értéke:  0xbfa8d2f4
*pn értéke: -1079454960
```

- Konstansok:

```
int n=5;
const int *pn = &n;
```

esetén *pn értéke konstans, vagyis ez nem változtatható, de n értéke igen.

```
int c=5;
int *const pc=&c;
```

esetén pc nem változtatható, de *pc igen.

Tömb és mutató

- A tömb neve önmagában, index nélkül mutatóként viselkedik (ld. scanf karakterlánc beolvasásánál).

~/info2/tombmutato.c

```
#include <stdio.h>
int main(void)
{
  int i=0;
  char t[] = "http://wiki.math.bme.hukarakterlanc"http://wiki.math.bme.hu;
  char *p = &t[0];
  printf("http://wiki.math.bme.hua tömb els? elemének címe: %p\n"http://wiki.math.bme.hu, p);
  printf("http://wiki.math.bme.hua tömb címe              : %p\n"http://wiki.math.bme.hu, t);
  for(; t[i]; i++)
    printf("http://wiki.math.bme.hu%p címen: %c\n"http://wiki.math.bme.hu, p+i, *(p+i) );

  return 0;
}
```

ugyanez egészekkel: ~/info2/tombmutato1.c

```
#include <stdio.h>
int main(void)
{
  int i=0, t[3]={11,12,13};
  int *p = &t[0];
  printf("http://wiki.math.bme.hua tömb els? elemének címe: %p\n"http://wiki.math.bme.hu, p);
  printf("http://wiki.math.bme.hua tömb címe              : %p\n"http://wiki.math.bme.hu, t);
  for(; i<3; i++)
    printf("http://wiki.math.bme.hu%p címen: %d\n"http://wiki.math.bme.hu, p+i, *(p+i) );

  return 0;
}
```

- a fenti példák alapján tehát: a t tömb t[i] eleme megegyezik *(p+i) vagy *(t+i) értékével, míg a cím, ahol van, a p+i vagy t+i vagy &t[i].
- egy többdimenziós (pl. t[3][3]) tömb esetén t, t[0] és &t[0][0] mutatók megegyeznek.

~/info2/tombmutato2.c

```
#include <stdio.h>
int main(void)
{
  int i, j, t[2][2] = { {1,2}, {3,4} };

  for(i=0; i<2; i++) {
    for(j=0; j<2; j++) {
      printf("http://wiki.math.bme.hu %d"http://wiki.math.bme.hu, *(*t + 2*i + j));
    }
    printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
  }

  for(i=0; i<4; i++)
    printf("http://wiki.math.bme.hu %d"http://wiki.math.bme.hu, *(*t + i));
  printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
  return 0;
}
```

Az alábbi ábra a t tömb és a t illetve t[0] és t[1] mutatók m?ködését szemlélteti. --> a mutatóból a mutatott helyre mutat, az = az adott címen lév? tartalom ekvivalens megadásait jelenti:

```
t --> t[0] --> t[0][0] = *t[0] = **t                    t[0][1] = *(t[0]+1) = *(*t+1)
      t[1] --> t[1][0] = *(t[0]+2) = *(*t+2) = *t[1]    t[1][1] = *(t[0]+3) = *(*t+3) = *(t[1]+1)
```

5. gyakorlat

1. Feladat (Vigenère-titkosítás): Egy olyan szöveget titkosítunk, melyben csak az angol ábécé bet?i szerepelnek A-tól Z-ig. A kulcs is csak e bet?ket tartalmazza. Képzeletben a nyílt szöveg alá írjuk a kulcsot. Ha a kulcs rövidebb, többször is leírjuk egymás után. Ezután a nyílt szöveg minden bet?je helyébe az a bet? kerül a titkos szövegbe, amelyik annyival van hátrébb az ábécében, amennyi az alatta lév? bet? sorszáma. A sorszámozást 0-val kezdjük, tehát A, B, C,..., Z sorszáma 0, 1, 2,..., 25. A „hátrébb az ábécében"http://wiki.math.bme.hu úgy értend?, hogy a Z után ciklikusan A következik. Például

```
A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Nyílt szöveg:    T  I  T  K  O  S  I  T  A  N  D  O ...
Bet? kódja:     19  8 19 10 14 19  8 19  0 13  3 14 ...
Kulcs (ZZABB):   Z  Z  A  B  B  Z  Z  A  B  B  Z  Z  A  B  B
```

```
Kulcs kódja:    25 25  0  1  1 25 25  0  1  1 25 25  0  1  1

Titkos kódja:   18  7 19 11 15 18  7 19  1 14  2 13 ...
Titkos szöveg:   S  H  T  L  P  S  H  T  B  O  C  N ...
```

```
#include <stdio.h>

int hossz(??? szoveg)
{
 ...
}

void vigenere(??? nyilt, ??? kulcs, ??? titkos)
{
 ...
}

int main(void)
{
  char nyilt[] ="http://wiki.math.bme.huEZATITKOSITANDOSZOVEGAZAZANYILTSZOVEG"http://wiki.math.bme
  char kulcs[] ="http://wiki.math.bme.huZZABB"http://wiki.math.bme.hu;
  char titkos[]="http://wiki.math.bme.huxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"http://wiki.math.bme
  vigenere( ??? );
  printf(titkos);
  printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
  return 0;
}
```

A megoldás:

```
#include <stdio.h>

int hossz(char *szoveg)
     // vagy int hossz(char szoveg[])
{
  int i=0;
  while(szoveg[i]) i++;
  return i;
}

void vigenere(char *nyilt, char *kulcs, char *titkos)
     // vagy void vigenere(char nyilt[], char kulcs[], char titkos[])
{
  int n,j,k;
  n=hossz(nyilt);
  k=hossz(kulcs);
  for(j=0; j<n; j++)
    titkos[j]=(nyilt[j]-'A'+kulcs[j%k]-'A')%('Z'-'A'+1) + 'A';
}

int main(void)
{
  char nyilt[] ="http://wiki.math.bme.huEZATITKOSITANDOSZOVEGAZAZANYILTSZOVEG"http://wiki.math.bme
  char kulcs[] ="http://wiki.math.bme.huZZABB"http://wiki.math.bme.hu;
  char titkos[]="http://wiki.math.bme.huxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"http://wiki.math.bme
  vigenere(nyilt, kulcs, titkos);
  printf(titkos);
  printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
  return 0;
}
```

2. Feladat: Írjunk egy függvényt, mely megkeresi egy mátrix legnagyobb elemét. A függvény els? paraméterében a mátrix, második és harmadik paraméterében a vizsgálandó mátrix sorainak és az oszlopainak száma szerepeljen. A memóriában a mátrix egy 10x10-es mátrix részmátrixaként szerepel.

```
#include <stdio.h>

#define SOROKSZ 10
#define OSZLOPOKSZ 10

int legnagyobb(int a[][OSZLOPOKSZ], int ssz, int osz);

int main()
{
  int t[SOROKSZ][OSZLOPOKSZ] = { {21, 22, 13},
                                 {35, 20, 15} };
  ...
  m = legnagyobb( t, ssz, osz);

  printf("http://wiki.math.bme.huA legnagyobb érték %d.\n"http://wiki.math.bme.hu, m);
  return 0;
}

int legnagyobb(int a[][OSZLOPOKSZ], int ssz, int osz)
{
  int i, j;
  int mx = a[0][0];

  ...

  return mx;
}
```

A megoldás:

```
#include <stdio.h>

#define SOROKSZ 10
#define OSZLOPOKSZ 10

int legnagyobb(int a[][OSZLOPOKSZ], int ssz, int osz);

int main()
{
  int t[SOROKSZ][OSZLOPOKSZ] = { {21, 22, 13},
                                 {35, 20, 15} };
  int ssz = 2;
  int osz = 3;
  int m;

  m = legnagyobb( t, ssz, osz);

  printf("http://wiki.math.bme.huA legnagyobb érték %d.\n"http://wiki.math.bme.hu, m);
  return 0;
}

int legnagyobb(int a[][OSZLOPOKSZ], int ssz, int osz)
{
  int i, j;
  int mx = a[0][0];
```

```
  for( i = 0; i < ssz; i++)
    for( j = 0; j < osz; j++)
      if ( a[i][j] > mx)
        mx = a[i][j];
  return mx;
}
```

5. el?adás (2008-03-14)

Tömbök és függvények átadása paraméterként (csak mutatókkal)

Index jelölés

ld. a gyakorlaton: ~/info2/mx.c

Mutató jelölés, típuskényszerítés (casting)

~/info2/mx2.c

```
#include <stdio.h>

#define SOROKSZ 10
#define OSZLOPOKSZ 10

int legnagyobb(int *a, int oszsz, int ssz, int osz)
{
  int i, j;
  int mx = *a;

  for( i = 0; i < ssz; i++)
    for( j = 0; j < osz; j++)
      if (  *(a + i*oszsz + j) > mx)
        mx =  *(a + i*oszsz + j);
  return mx;
}

int main()
{
  int t[SOROKSZ][OSZLOPOKSZ] = { {21, 42, 13},
                                 {35, 20, 15} };
  int ssz = 2;
  int osz = 3;
  int m;

  m = legnagyobb( (int *)t, OSZLOPOKSZ, ssz, osz);

  printf("http://wiki.math.bme.huA legnagyobb érték %d.\n"http://wiki.math.bme.hu, m);
  return 0;
}
```

Függvény átadása

Függvénymutató deklarálása:

```
int (*mutfv)( int )
```

Ez így még nem mutat sehová, csak egy mutatót deklarál, mely csak olyan függvényre mutathat majd, amelynek egyetlen egész argumentuma van, és amely egy egészt ad vissza.

A zárójel kell, mert

```
int *mutfv( int )
```

olyan függvényt deklarálna, mely egészre mutató mutatót ad vissza.

~/info2/fvmut.c

```
#include <stdio.h>

int ossz(int, int);         // fv prototípusok
int szor(int, int);

int main(void)
{
  int a = 4;
  int b = 3;
  int e = 0;                // eredmény
  int (*fvm)(int, int);     // függvény mutató deklaráció

  fvm = ossz;               // az ossz() függvényre mutat
  e = fvm(a, b);            // ossz(a, b) meghívása
  printf("http://wiki.math.bme.huAz összeg = %d\n"http://wiki.math.bme.hu, e);

  fvm = szor;
  e = fvm(a, b);
  printf("http://wiki.math.bme.huA szorzat = %d\n"http://wiki.math.bme.hu, e);

  return 0;
}

int ossz(int x, int y) {
  return x + y;
}

int szor(int x, int y) {
  return x * y;
}
```

Véletlen sorozat generálása

Véletlen számok generálása: rand() az stdlib.h betöltése után egy [0,RAND_MAX] = [0,INT_MAX] intervallumba es? egészt generál. *random seed*: srand(<*szám*>) hívással befolyásolható az els?nek generált -- és ezzel az összes -- szám, mivel ez csak álvéletlen sorozat. Valódi véletlenség látszatához használjuk a time(NULL) függvényt.

~/info2/rand.c

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

int main (void) {
  int i;
```

```
  srand( (unsigned int) time( NULL ));         // seed
  for (i=0; i<5; ++i) printf("http://wiki.math.bme.hu%d "http://wiki.math.bme.hu,rand()%64); // [0
  printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);

  return 0;
}
```

Struktúra

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

int main(void)
{
  struct allat                  // struktúra deklaráció
  {
    char nev[20];
    int db;
    struct allat *kov;          // mutató a következ?re
  };

  struct allat *elso = NULL;    // mutató az els? állatra
  struct allat *ez   = NULL;    // mutató erre az állatra
  struct allat *utso = NULL;    // mutató az el?z?re

  char c = '\0';

  for( ; ; )
  {
    printf("http://wiki.math.bme.hu\nBeviszed egy allat adatait? (i/n) "http://wiki.math.bme.hu);

    scanf("http://wiki.math.bme.hu %c"http://wiki.math.bme.hu,&c);
    if(c == 'n' || c == 'N') break;

    // memóriaallokáció
    ez = (struct allat*) malloc(sizeof(struct allat));

    if(elso == NULL) elso = ez; // mutató az els? állatra

    if(utso!= NULL) utso -> kov = ez;

    printf("http://wiki.math.bme.hu\nAz allat neve? "http://wiki.math.bme.hu);
    scanf("http://wiki.math.bme.hu%s"http://wiki.math.bme.hu, ez -> nev);

    printf("http://wiki.math.bme.hu\nHany %s lesz a hajon? "http://wiki.math.bme.hu, ez -> nev);
    scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu, &ez -> db);

    ez->kov = NULL;             // lehet, hogy ez az utolsó
    utso = ez;                  // áttesszük az utso-ba
  }

                               // állatok listájának kiírása
  ez = elso;

  while (ez != NULL)            // amíg ez egy érvényes mutató
  {
    printf("http://wiki.math.bme.hu\n%d darab %s,\n"http://wiki.math.bme.hu, ez->db, ez->nev);
    utso = ez;                  // mentés memóriafelszabadításhoz
    ez = ez->kov;               // mutató a következ?re
    free(utso);                 // memóriafelszabadítás
  }
  return 0;
```

```
}
```

Struktúra átadása függvényértékként

~/info2/noe1.c

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

struct allat *uj_allat(void);

struct allat                  // struktúra deklaráció
{
  char nev[20];
  int db;
  struct allat *elozo;        // mutató az el?z?re
  struct allat *kov;          // mutató a következ?re
};

int main(void)
{
  struct allat *elso = NULL;    // mutató az els? állatra
  struct allat *ez   = NULL;    // mutató erre az állatra
  struct allat *utso = NULL;    // mutató az el?z?re
  char c = '\0';

  for(;;)
  {
    printf("http://wiki.math.bme.hu\nBeviszed egy allat adatait? (i/n) "http://wiki.math.bme.hu);
    scanf("http://wiki.math.bme.hu %c"http://wiki.math.bme.hu,&c);
    if(c == 'n' || c == 'N') break;

    // memóriaallokáció, adatbevitel
    ez = uj_allat();

    if(elso == NULL)
    {
      elso = ez;
      utso = ez;
    }
    else
    {
      utso->kov = ez;
      ez->elozo = utso;
      utso = ez;
    }
  }

  /* KIÍRÁS */
  ez = elso;
  while (ez != NULL)            // amíg ez egy érvényes mutató
  {
    printf("http://wiki.math.bme.hu%d darab %s\n"http://wiki.math.bme.hu, ez->db, ez->nev);
    utso = ez;                  // mentés memóriafelszabadításhoz
    ez = ez->kov;               // mutató a következ?re
    free(utso);                 // memóriafelszabadítás
  }
  return 0;
}
```

```
struct allat *uj_allat(void)
{
  struct allat *a;

  a = (struct allat*) malloc(sizeof(struct allat));

  printf("http://wiki.math.bme.hu\nAz allat neve? "http://wiki.math.bme.hu);
  scanf("http://wiki.math.bme.hu%s"http://wiki.math.bme.hu, a -> nev);

  printf("http://wiki.math.bme.hu\nHany %s lesz a hajon? "http://wiki.math.bme.hu, a -> nev);
  scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu, &a -> db);

  a->kov = a->elozo = NULL;

  return a;
}
```

6. gyakorlat (2008-03-18, 2008-03-21)

1. feladat: Az el?adáson szerepl? duplán láncolt lista kódját írjuk át úgy, hogy mindkét irányba írjuk ki az állatok listáját, és a -> jelölést cseréljük le vele ekvivalensre.

2. feladat: Az óvódai évbúcsúztatón kiszámolással döntik el, hogy a csoportból ki lesz az a 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 (~/info2/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 (~/info2/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<n-2; i++) {                  // minden kies? gyerekre
      for(j=0; j<k-1; j++) akt=(*akt).kov; // vagy akt = akt->kov
      printf("http://wiki.math.bme.hu%d "http://wiki.math.bme.hu, (*(*akt).kov).szam);   // vagy (
      (*akt).kov=(*(*akt).kov).kov;         // akt->kov = (akt->kov)->kov
    }
    printf("http://wiki.math.bme.hu; %d %d\n"http://wiki.math.bme.hu, (*(*akt).kov).szam, (*(*(*ak
                //vagy printf("http://wiki.math.bme.hu; %d %d\n"http://wiki.math.bme.hu, (akt->kov)
  }
  return 0;
}
```

6. el?adás (2008-03-21)

Struktúrák igazítása

A 2/4/8 hosszú adatok (short/int/double/...) csak 2/4/8-cal osztható címen kezd?dhetnek.

~/info2/igazit.c

```
/* megnézzük, hogy a struktúrában hogy történik az igazítás */
#include <stdio.h>

int main(void) {
  struct igazito {
    int a;
    char b;
    double d;
    char f;
    struct igazito *p;
  } ig={1, 'w', 3.4, 'f', &ig};

  printf("http://wiki.math.bme.hu\n%d %p, %c %p, %f %p, %c %p\n%p %p\n"http://wiki.math.bme.hu,
    ig.a, &ig.a, ig.b, &ig.b, ig.d, &ig.d, ig.f, &ig.f, ig.p, &ig.p);
  return 0;
}
```

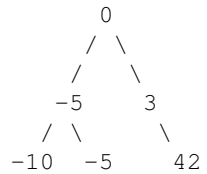
Bináris keres?fák

Bináris fa

- 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
- részfa gyökere

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:



```
   / \
  -5  -2
```

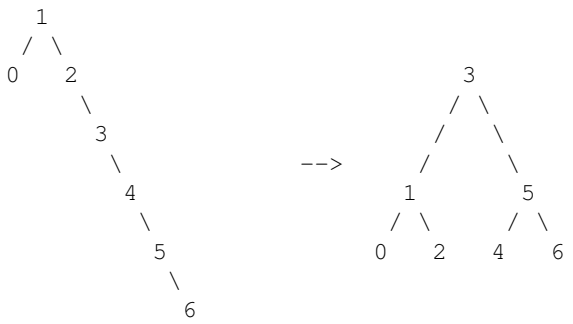
Alapm?veletek bináris keres?fákkal:

- 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.

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 ($O(n)$).

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:

```
#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;
}
```

Célunk, hogy meg tudjuk írni a bináris keres?fába való beszúrás algoritmusát.

A bináris keres?fába való beszúrás 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:

~/info2/fabeolv.c

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

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;
}

void kirajzol(struct csucs *gyoker, int nszokoz) {
  int i;
  for (i=nszokoz; i>0; i--) 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);
}

int main(void) {
  int i, csucsoksz=0;
  struct csucs *gyoker=NULL;
  struct csucs *fa;

  printf("http://wiki.math.bme.huHány csúcsa lesz a fának? "http://wiki.math.bme.hu);
  scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu, &csucsoksz);
  fa = (struct csucs*) malloc(csucsoksz*sizeof(struct csucs));
  if(fa == NULL) {
    printf("http://wiki.math.bme.huNincs eleg memoria!"http://wiki.math.bme.hu);
    return 0;
  }
  for(i=0; i<csucsoksz; i++) {
    printf("http://wiki.math.bme.hu%d. csúcs értéke? "http://wiki.math.bme.hu, i+1);
    scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu,&fa[i].ertek);
    beszur(&gyoker, &fa[i]);
  }
  kirajzol(gyoker,1);
  return 0;
}
```

7. gyakorlat (2008-03-25, 2008-03-28)

Feladat: Egy tone üzemmódú telefonos menürendszert tervezünk, mely két kívánalomnak kell hogy eleget tegyen.

1. egy szám felhívása után az ügyfél mindig olyan kérédseket kap, melyekre pontosan 2 lehet?ség közül kell választania;
2. a kezd?ponttól a kívánt menüpontig való eljutáshoz szükséges gombnyomások számának várható értéke a lehet? legkisebb legyen. Ehhez az el?z? év adataiból meg is becsülték az egyes menüpontok népszer?ségét.

A feladat tehát az, hogy ismerve a menüpontok számát és relatív népszer?ségét, alakítsunk ki megfelel? menürendszert, és megvalósítására írjunk programot. A bemenet minden sorában a menüpontok száma és az egyes menüpontok relatív népszer?sége olvasható, például

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

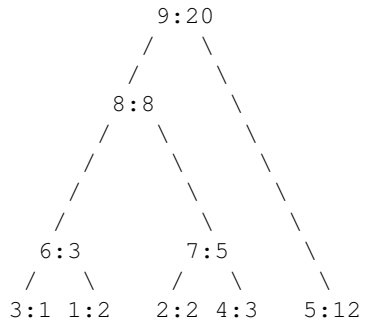
azt jelenti, hogy 5 menüpont van, melyekre annak valószn?sége, hogy egy telefonáló épp az adott menüpontot keresi, az 1. menüpont esetén 2/20, a 2. menüpont esetén 2/20, a 3. menüpont esetén 1/20, a 4. menüpont esetén 3/20 és az 5. menüpont esetén 12/20. Feltételezhet?, hogy kevesebb, mint egymillió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

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

A fenti kimenetben pl. a 7. helyen álló `7:5,8,2,4` 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. (ld. http://en.wikipedia.org/wiki/Huffman_coding)

A megírandó program (~/info2/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 fa kiírása */
void kiir(void) {
  int i;
  for (i=1; i<=n; i++) {
    printf("http://wiki.math.bme.hu%d:%d,%d,%d,%d "http://wiki.math.bme.hu, i,
           t[i].nepszeruseg,
           (t[i].szulo == NULL) ? 0 : ... ,
           (t[i].bal   == NULL) ? 0 : ... ,
           (t[i].jobb  == NULL) ? 0 : ... ;
  }
  printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
}

/** 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++;
      ...
    }

    kiir();

  }
  return 0;
}
```

A program egy lehetséges változata:

```
#include <stdio.h>

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

#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 n (t[1] ... t[n]) */
int n;

void kiir(void) {
  int i;
  for (i=1; i<=n; i++) { /* a fa kiírása */
    printf("http://wiki.math.bme.hu%d:%d,%d,%d,%d "http://wiki.math.bme.hu, i,
           t[i].nepszeruseg,
           (t[i].szulo == NULL) ? 0 : t[i].szulo->sorszam,
           (t[i].bal   == NULL) ? 0 : t[i].bal->sorszam,
           (t[i].jobb  == NULL) ? 0 : t[i].jobb->sorszam);
  }
  printf("http://wiki.math.bme.hu\n"http://wiki.math.bme.hu);
}

/* 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) {
  int i;

  *bp = *jp = &t[0];
  for( i=1; i<n+1; i++ ) {
    if( t[i].szulo ) continue;  /* ha van szül?je, kihagyjuk */
    if( t[i].nepszeruseg < (*bp)->nepszeruseg ) {
      *jp = *bp;
      *bp = &t[i];}
    else if( t[i].nepszeruseg < (*jp)->nepszeruseg ) {
      *jp = &t[i];
    }
  }
}

int main(void) {
  int i, m;
  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)) {
    m=n;
```

```
    for(i=1; i<=n; i++) { /* bementi népszer?ségek olvasása */
      t[i].sorszam=i;
      scanf("http://wiki.math.bme.hu%d"http://wiki.math.bme.hu, &t[i].nepszeruseg);
      t[i].bal = t[i].jobb = t[i].szulo = NULL;
    }

    for(i=1; i<m; i++) { /* m-1 db összekötés */
      ket_legkisebbet_keres(&b, &j);
      n++;
      t[n].nepszeruseg = b->nepszeruseg + j->nepszeruseg;
      t[n].sorszam = n;
      t[n].szulo = NULL;
      t[n].bal = b;
      t[n].jobb = j;
      b->szulo = &t[n];
      j->szulo = &t[n];
    }
    kiir();
  }
  return 0;
}
```

7. el?adás (2008-03-28)

Ami az eddigiekb?l kimaradt

Felsorolások (enumerátorok), az enum adattípus

`enum` *típusazonosító* {*felsorolás*} *változók*;

```
enum hetkoznap {hetfo, kedd, szerda, csutortok, pentek};
enum Boolean {true, false} elment=false;
...
enum hetkoznap fogorvos=szerda;
fogorvos=pentek;
...
elment=true;
```

Automatikusan 0-tól sorszámozódó int típusok. Az érték megválasztható:

```
enum t_vegu_szamjegyek {ot=5, hat, het};
```

típusazonosító nélküli *egyszer használatos* felsorolások:

```
enum {tegnap, ma, holnap} mikor=ma;
...
mikor=holnap;
```

typedef -- típus definiálása

Definiáljuk a Gauss-egészek struktúráját:

```
struct Gauss_egesz {
  int x;
  int y;
} a, b;
```

Legyen egy ilyen típusunk:

```
typedef struct Gauss_egesz GEtipus;
GEtipus a, b;
```

vagy összeolvasztva:

```
typedef struct Gauss_egesz {
  int x;
  int y;
} GEtipus;
GEtipus a, b;
```

esetszétválasztás (switch/case/default)

Az if-else csak kétfelé elágazást enged, a switch többfelé:

```
switch(ch) {
  case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '
    printf("http://wiki.math.bme.hudecimális"http://wiki.math.bme.hu);
    break;
  case 'a': case 'b': case 'c': case 'd': case 'e': case 'f':
    printf("http://wiki.math.bme.huhexadecimális"http://wiki.math.bme.hu);
    break;
  default:
    print("http://wiki.math.bme.hunem számjegy"http://wiki.math.bme.hu);
}
```

Feltételes kifejezés

a feltételes m?velet *ternér m?velet*, azaz 3 argumentuma van.

feltétel ? *kifejezés_1* : *kifejezés_2*

Például:

```
z = x<y ? x : y;
```

Ami a következ? kóddal ekvivalens:

```
if(x<y)
  z = x;
else
  z = y;
```

Állapottér, állapotgép (automata)

Ld. Szeberényi Imre egyik vagy másik anyagában.

8. gyakorlat (2008-04-01, 2008-04-04)

Feladat: 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 van megadva. 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 (~/info2/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 (~/info2/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 */

  /* 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+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;
}
```