Prog1, C tárgyalás

Magasszintű programozási nyelvek 1
BSc előadás

Dr. Bátfai Norbert
egyetemi adjunktus
http://www.inf.unideb.hu/~nbatfai/
Debreceni Egyetem, Informatikai Kar,
Információ Technológia Tanszék
batfai.norbert@inf.unideb.hu
Skype: batfai.norbert

Prog1_2.ppt, v.: 0.0.7, 2012. 03. 06.
http://www.inf.unideb.hu/~nbatfai/#p1
http://nehogy.fw.hu/

Az óra blogja: http://progpater.blog.hu/
A Nokia Ovi store-ban is elérhető: http://store.ovi.com/content/100794

Felhasználási engedély

Bátfai Norbert
Debreceni Egyetem, Informatikai Kar, Információ Technológia Tanszék
<nbatfai@inf.unideb.hu, nbatfai gmail com>

Célok és tartalom

Előadás

a) Kolmogorov bonyolultság, véletlen számsorozat
b) Élettartam és hatáskör (érvényességi tartomány)
c) Mutatók és több dimenziós tömbök, mutatóaritmetika, dinamikus tárkezelés.

Labor

a) Egy saját PageRank implementáció: http://progpater.blog.hu/2011/02/13/bearazzuk_a_masodik_labort
b) EXOR törés: http://progpater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia
c) Hatáskör „megbolondítása” a fork rendszerhívással: http://progpater.blog.hu/2011/02/07/zombiveszely_1
d) a crashme.c forrásának a laborvezető által celebrált átnézése (csak a jelkezelés szempontjából lényegi működés)
e) A PP 36-40 oldal példáinak letöltése, kipróbálása, megbeszélése a laborvezetővel (fork rendszerhívás és társai).

Laborkártyák

a) Forrás, mutatós és operátoros kártyák

Otthoni opcionális feladat

a) A japán világbajnok HELIOS csapat szoftvereinek otthoni installálása (librcsc, agent2D, soccervindow stb.) http://progpater.blog.hu/2011/02/05/a_felkelo_nap_palyaja

Kapcsoldó videók, videómagyarázatok és blogok

1) http://progpater.blog.hu/2011/02/19/a_masodik_eloadas_fizikailag
   http://www.youtube.com/watch?v=iugf3e_n9-8
2) http://progpater.blog.hu/2011/02/13/bearazzuk_a_masodik_labort
3) http://progpater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia
4) http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

2012:

1) http://progpater.blog.hu/2012/02/15/retreat_hell
2) http://progpater.blog.hu/2012/02/25/de_ik_prog_labor_labdarugo_bajnoksag_es_kupa

Az írásbeli és a szóbeli vizsgán bármi (jegyzet, könyv, forráskód, számítógép mobiltelefon stb.) használható! (Az írásbeli vizsgán beszélni viszont tilos.) Hiszen az én feladatom az lesz, hogy eldöntsem, jól felkészült programozóval, vagy mennyire felkészült programozóval állok szemben.

Minimális gyakorlati cél

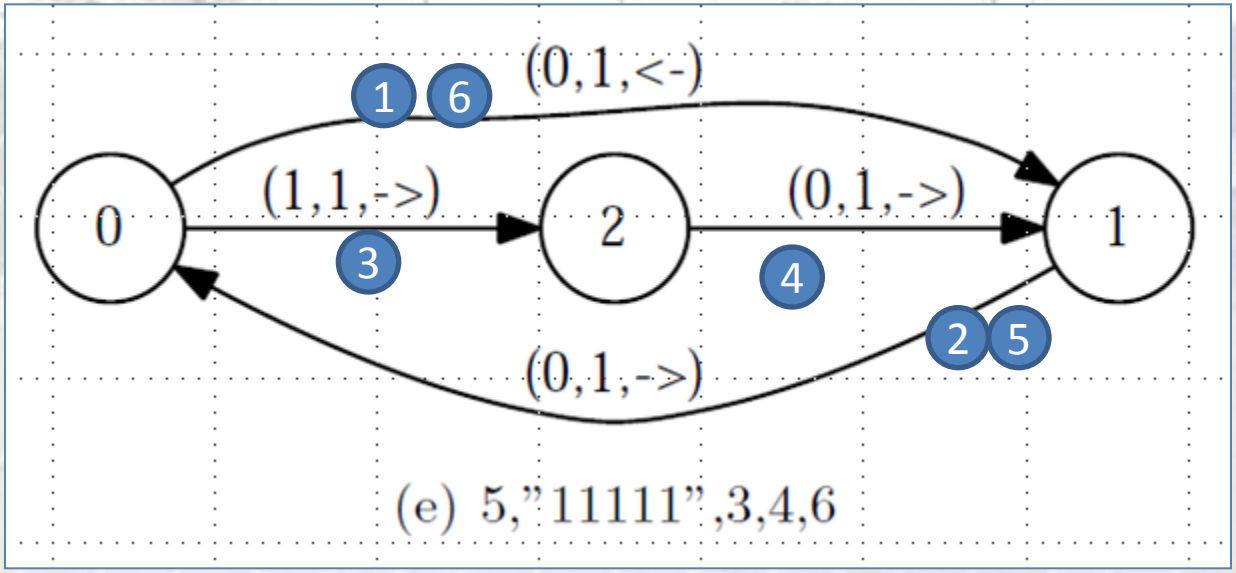
A hallgató meg tudjon írni (másolás alapján) és le tudjon fordítani kisebb C programokat, illetve önállóan ki tudjon egészíteni (egyszerű iterációval, elágazással, összeadással, szorzással (PageRank implementáció), bitenkénti logikai (EXOR törés) műveletekkel) forrásban adott algoritmust.

A malloc() könyvtári függvénnyel tudja dinamikusan kezelni a tárat egyszerű esetekben (ismerje a malloc, calloc, free könyvtári függvényeket). Továbbá ismerje a fork rendszerhívást!

Minimális elméleti cél

1) Tudjon mesélni olyan problémákról, amit nem lehet (a klasszikus értelemben vett számítógépes) programmal megoldani. Például megállási probléma (lásd az első előadást), Kolmogorov bonyolultság, esetleg a Radó Tibor féle szorgos hód probléma.
2) A Kolmogorov bonyolultság fogalmának felhasználásával meg tudja mutatni, hogy (a klasszikus értelemben vett számítógépes) programmal nem lehet véletlen számsorozatot generálni.
3) Hatáskör, élettartam fogalma, lyuk a hatáskörben jelensége.
4) C nyelv kapcsán: belső, külső és statikus változók, extern deklaráció, deklaráció és definíció különbözősége.
5) C nyelv kapcsán: mutatók, mutatók és tömbök, mutatók és több dimenziós tömbök, mutatóaritmetika.

A program tár- és idő bonyolultsága, illetve a program-méret bonyolultság



http://progpater.blog.hu/2011/02/06/a_forras_szovete_avagy_dest_src
Mi volt a kisbajnokság itt?

Kép forrása: http://arxiv.org/abs/0908.1159

Figure 1: "Placid Platypus machines" are found by our programs, (# of 1's, "$T(\lambda)$", # of states, # of rules, # of steps)

A program-méret bonyolultság

```
nbatfai@hpserver:~/p1> more elso.c
#include <stdio.h>

int
main(void)
{
  printf("0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000");
  return 0;
}
nbatfai@hpserver:~/p1> wc elso.c
   8    9 1066 elso.c
nbatfai@hpserver:~/p1>
```

A programhossz alapú bonyolultság intuitív bevezetése: tekintsünk néhány programot: a programok hosszát (lényegtelen, de most wc-vel számoljuk) és hogy milyen 1000 bites sztringet ír ki, ha futtatjuk!

Ugyanez Javában: Javát tanítok,
http://www.tankonyvtar.hu/main.php?objectID=5314387

A program-méret bonyolultság

```
nbatfai@hpserver:~/p1> more masodik.c
#include <stdio.h>

int
main(void)
{
  for(int i=0; i<1000; ++i)
    printf("0");
  return 0;
}

nbatfai@hpserver:~/p1> wc masodik.c
 10  13 102 masodik.c
```

A programhossz alapú bonyolultság intuitív bevezetése: tekintsünk néhány programot: a programok hosszát (lényegtelen, de most wc-vel számoljuk) és hogy milyen 1000 bites sztringet ír ki, ha futtatjuk!

A program-méret bonyolultság

```
hpserver:~/p1> more harmadik.c
#include <stdio.h>

int
main(void)
{
  for(int i=0; i<500; ++i)
    printf("01");
  return 0;
}

nbatfai@hpserver:~/p1> wc harmadik.c
 10  13 102 harmadik.c
nbatfai@hpserver:~/p1> ./harmadik
01010101010101010101010101010101010101010...
```

A programhossz alapú bonyolultság intuitív bevezetése: tekintsünk néhány programot: a programok hosszát (lényegtelen, de most wc-vel számoljuk) és hogy milyen 1000 bites sztringet ír ki, ha futtatjuk!

A program-méret bonyolultság

```
nbatfai@hpserver:~/p1> more negyedik.c
#include <stdio.h>

int
main(void)
{
  printf("0000000000");
  for(int i=0; i<980; ++i)
    printf("1");
  printf("0000000000");

  return 0;
}

nbatfai@hpserver:~/p1> wc negyedik.c
 13  15 162 negyedik.c
nbatfai@hpserver:~/p1> ./negyedik
00000000001111111111111111111111111...
```

A programhossz alapú bonyolultság intuitív bevezetése: tekintsünk néhány programot: a programok hosszát (lényegtelen, de most wc-vel számoljuk) és hogy milyen 1000 bites sztringet ír ki, ha futtatjuk!

A program-méret bonyolultság

```
nbatfai@hpserver:~/p1> more otodik.c
#include <stdio.h>

int
main(void)
{
  for(int i=0; i<44; ++i)
  {
    printf("0");
    for(int j=0; j<i; ++j)
      printf("1");
  }
  printf("1111111111");

  return 0;
}

nbatfai@hpserver:~/p1> wc otodik.c
 16  21 176 otodik.c
nbatfai@hpserver:~/p1> ./otodik
00101101110111011110111111011111101111111011111111110
```

ág intuitív bevezetése: tekintsünk néhány programot: a programok hosszát (lényegtelen, de most wc-vel számoljuk) és hogy milyen 1000 bites sztringet ír ki, ha futtatjuk!

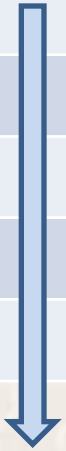
A program-méret bonyolultság

```
hpserver:~/p1> more hatodik.c
#include <stdio.h>

int
main(void)
{
  printf("0101101000001001000011100011111011110001010010011110100100000001010010011
101010011111110100000010010000110110001010111100100110101000100100000101100110010001
010110011001000110100111000011010111001111101101000110111000101001000000011100111100 0
101001001111100100111101101111000110111011010011110000101001010000111110000100011101
100111001010110000000110111110110111011011101101100001010101100110001001110011101011
000111101001111010000000111101010111000011011000011110010010101010000000111111101100
000100001000001000000011111000101010111111100000001000111110111010011010001011010011 1
110101001110101001101011010110111100110011010110110111011011110011011100010010101111
111111111100000101011110011010010100110111101100100010010010111011000111101110000110
010011010100110111101011001011101011011110110100100011100011111000101000101111111001
000100101110100000011100011010101010000011001001001001111000011011011100111111010110
101111010110001100110000101101010000111111101000001001110111010101001011010011100100
01");
  return 0;
}
nbatfai@hpserver:~/p1> wc hatodik.c
   8     9 1066 hatodik.c
```

programok hosszát (lényegtelen, de most wc-vel számoljuk) és hogy milyen 1000 bites sztringet ír ki, ha futtatjuk!

A program-méret bonyolultság

Sorozat	Program	Program-méret
0^{1000}	elso.c	1066
0^{1000}	masodik	102
$(01)^{500}$	3.	102
$0000000000(1)^{980}0000000000$	4.	162
$\Pi\ 0(1)^{i}$	5.	176
1000 db„véletlen”	6.	1066



Intuitíven erre nő a bonyolultság

A programhossz alapú bonyolultság intuitív bevezetése: tekintsünk néhány programot: a programok hosszát (lényegtelen, de most wc-vel számoljuk) és hogy milyen 1000 bites sztringet ír ki, ha futtatjuk!

A program-méret bonyolultság

```
nbatfai@hpserver:~/p1> more hetedik.c
#include <stdio.h>

int
main(void)
{
  int c;

  while((c=getchar()) != EOF)
    putchar(c);

    return 0;
}
nbatfai@hpserver:~/p1> wc hetedik.c
 12  14 112 hetedik.c
```

A programhossz alapú bonyolultság intuitív bevezetése: tekintsünk néhány programot: a programok hosszát (lényegtelen, de most wc-vel számoljuk) és hogy milyen 1000 bites sztringet ír ki, ha futtatjuk!

A program-méret bonyolultság

```
nbatfai@hpserver:~/p1> echo "01011010000010010000111000111110111100010100100111101000
10000000101001000111010100111111101000000100100001101100010101111001001101010001001
00001011001100100010101100110010001101001110000110101110011111101101000110111000101
00000011110011110001010010011111001001111011011110001101110110100111110000101001010
01111100001000111011001110010101100000001101111101101110111011011000010101011001100
00010011100111010110001111010011111010000001111010101110000110110000011110010010101
00000011111111101100000100001000001000000011111000101010111110000000100011111011010
01101000101101001111101010011101010011010110101101111001100110101101101110110111001
10111000100101011111111111110000010101111001101001010011011110110010001001001011101
10001111011100001100100110101001101111010110010111010110111101101001000111000111100
01010001011111110010001001011101000000111000110101010100000110010010010011110000110
10111001111110101101011110101100011001100001011010100001111111010000010011101110101
10010110100111001000 1"|./hetedik
01011010000010010000111000111110111100010100100111101001000000010100100011101010011
11110100000010010000110110001010111100100110101000100100000101100110010001010110011
01000110100111000011010111001111101101000110111000101001000000111001111000101001001
11100100111101101111000110111011010011111000010100101000011111000010001110110011100
10110000000110111110110111011011101101100001010101100110001001110011110101100011110
01111010000001111010101110000110110000111100100101010100000001111111101100000100001
00001000000011111000101010111111000000010001111101110100110100010110100111110101001
10101001101011010110111100110011010110110111011011110011011100010010101111111111111
00000101011110011010010100110111101100100010010010111011000111101110000110010011010
00110111101011001011101011011111011010001110001111100010100010111111100100010010111
10100000011100011010101010000011001001001001111000011011011100111111010110101111010
10001100110000101101010000111111101000001001110111010101001011010011100100 01
```

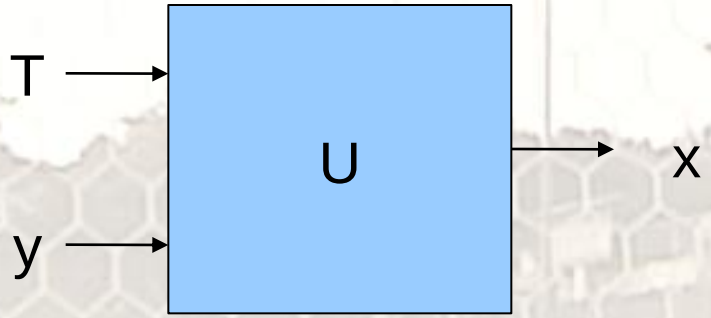
A programhossz alapú bonyolultság intuitív bevezetése: tekintsünk néhány programot: a programok hosszát (lényegtelen, de most wc-vel számoljuk) és hogy milyen 1000 bites sztringet ír ki, ha futtatjuk!

A program-méret bonyolultság

Sorozat	Program kód	Program inputja	Program-méret	A bonyolultság felső becslése
0^{1000}	elso.c	0	1066	1066
0^{1000}	masodik	0	102	102
$(01)^{500}$	3.	0	102	102
$0000000000(1)^{980}0000000000$	4.	0	162	162
$\Pi\ 0(1)^{i}$	5.	0	176	176
1000 db„véletlen”	6.	0	1066	1066
„cat” jellegű	7.	1000	112	1112

A programhossz alapú bonyolultság intuitív bevezetése: tekintsünk néhány programot: a programok hosszát (lényegtelen, de most wc-vel számoljuk) és hogy milyen 1000 bites sztringet ír ki, ha futtatjuk!

A Solomonoff-Chaitin-Kolmogorov bonyolultság



$$C(x) = min\{|Ty| : U(T,y) = x\}$$

Invariancia tétel (nem lényeges, hogy milyen U géppel dolgozunk),

Használják valahol ezt a fogalmat?

Algoritmikus információs távolság
1998, Information Distance
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.9275

$$E(x,y) = min\{|T| : U(T,y) = x, U(T,x) = y\}$$

Hasonlósági metrika
2003, The similarity metric
http://arxiv.org/abs/cs.CC/0111054
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1.3346

$$d(x,y) = \frac{E(x,y)}{max\{C(x),C(y)\}}$$

(normalizált információs távolság)

Használják valahol ezt a fogalmat?

NAME

ncd - compute the Normalized Compression Distance

SYNOPSIS

ncd [-c compressor] [-o filename] [-bcdhLnqsv] [-o filestem] [-d|f|l|p|t string] ... [arg1] [arg2]

DESCRIPTION

The Normalized Compression Distance between two objects is defined as

NCD(a,b) = (C(a,b) - min(C(a),C(b))) / max(C(a),C(b))

where

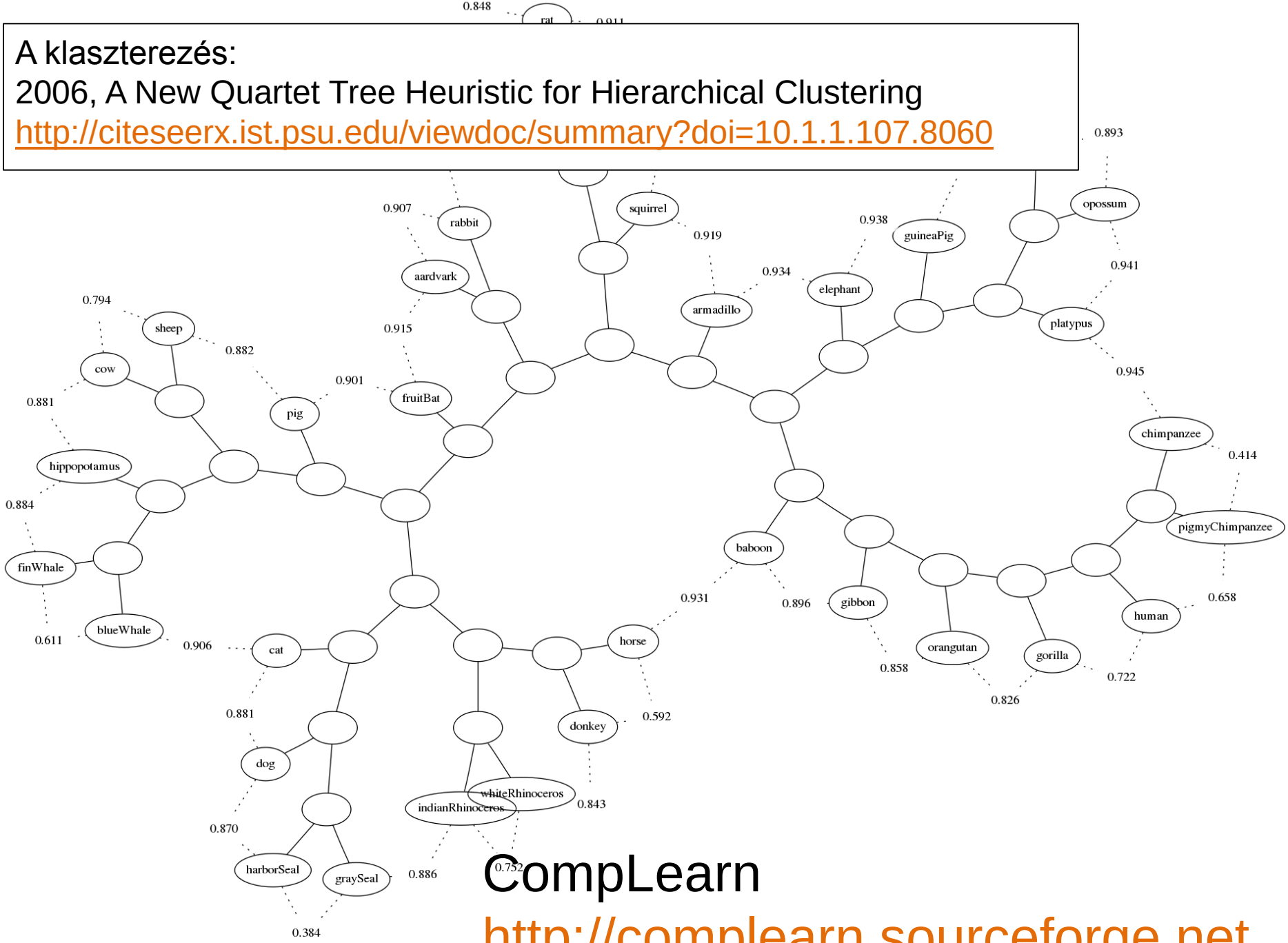
C(a,b) means "the compressed size of the concatenation of a and b"

C(a) means "the compressed size of a"

C(b) means "the compressed size of b"

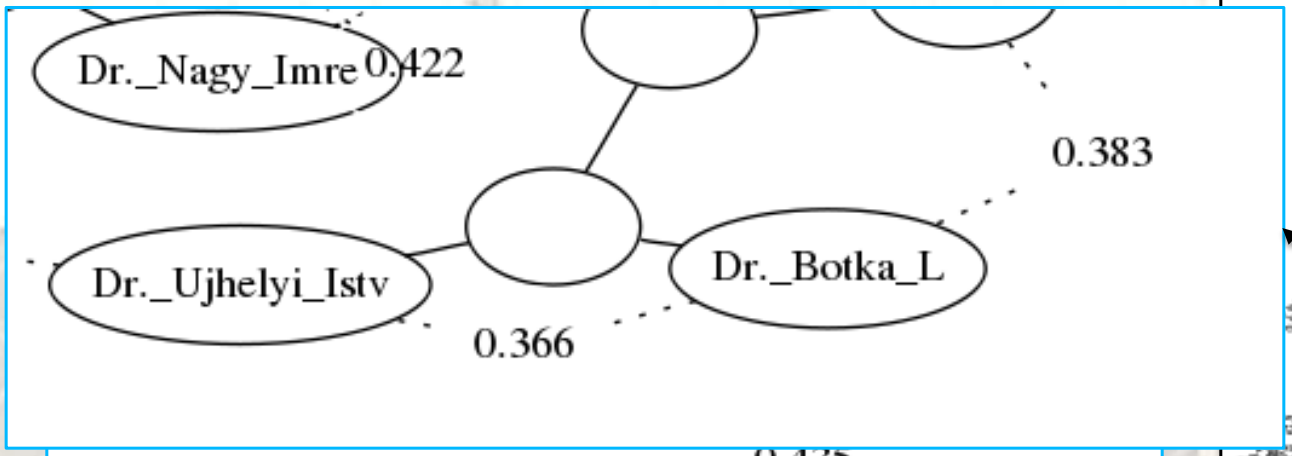
ncd will print a non-negative number (typically, but not always, 0 <= x < 1.1) representing how different the two objects are. Smaller numbers

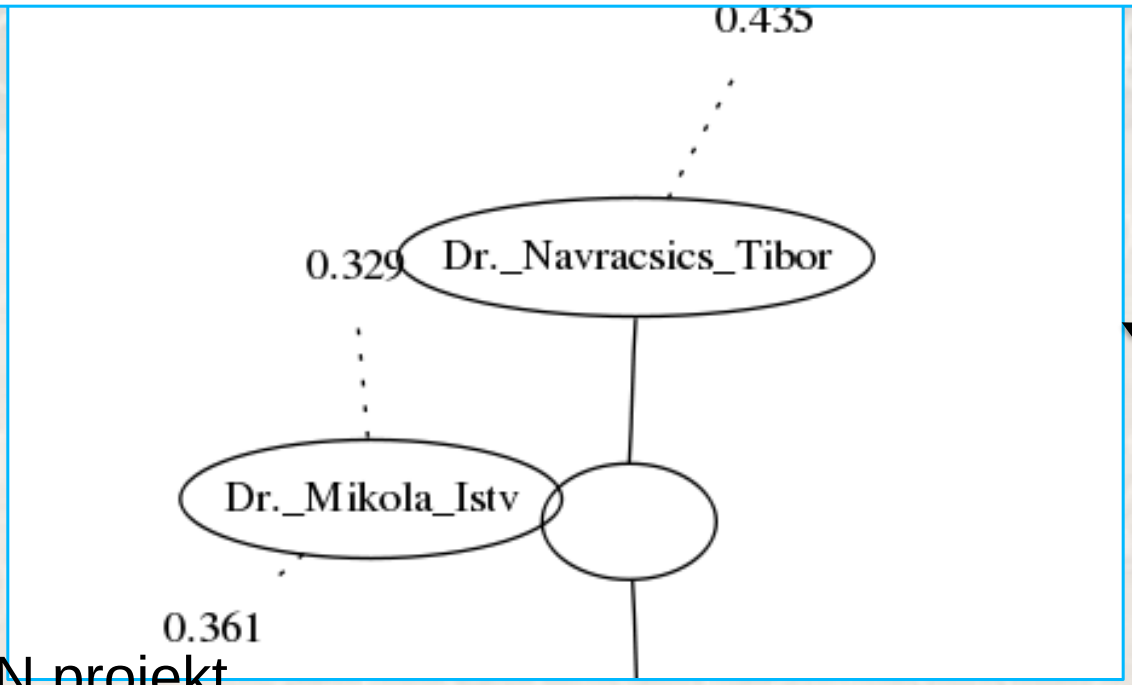
A klaszterezés:
2006, A New Quartet Tree Heuristic for Hierarchical Clustering
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.107.8060

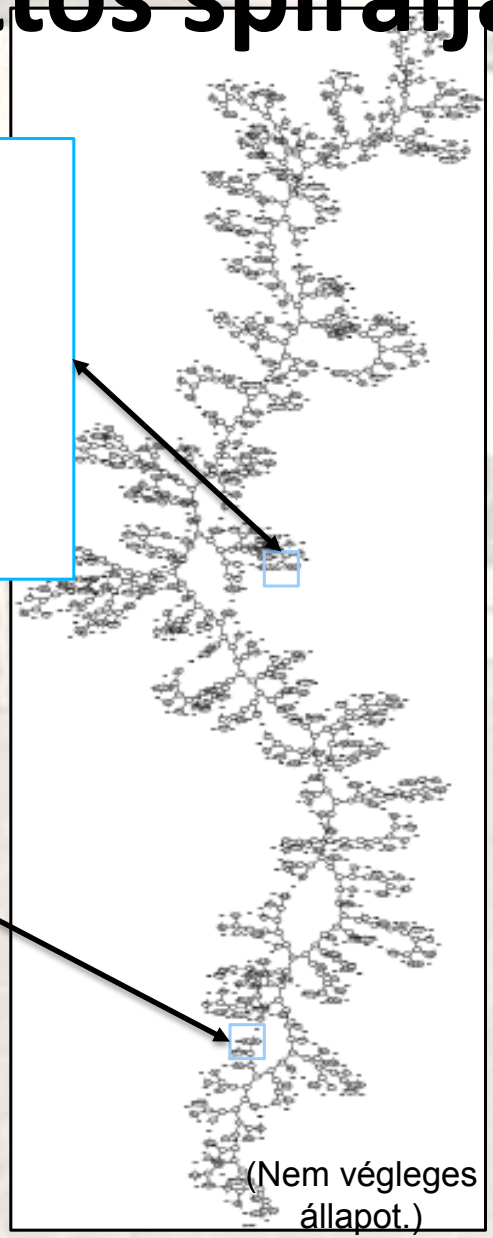


CompLearn
http://complearn.sourceforge.net

CompLearn „az ország kettős spirálja”







(DDN projekt, http://www.inf.unideb.hu/~nbatfai/ddn/ddnevkonyvdb.pdf

Nem kiszámítható!

```c
#include <stdio.h>

#define BONYOLULTSAG_FGV_HOSSZA 10000

extern int bonyolultsag(char *);
extern char * binarisba(int);

int
main(void)
{
  char * sorozat;
  int i=0;

  for(;;)
  {
     sorozat = binarisba(i++);
     if(bonyolultsag(sorozat) > 10 *  BONYOLULTSAG_FGV_HOSSZA)
       break;
  }

  printf("%s", sorozat);

  return 0;
}
```

Kilép és kiírja a sorozatot.

sorozat

Lásd Javában: 1.8. példa - A Chaitin-Kolmogorov bonyolultság nem kiszámítható!
http://www.tankonyvtar.hu/main.php?objectID=5314387

Nem kiszámítható!

```
nbatfai@hpserver:~/p1> wc cbonyo.c
 24   38 345 cbonyo.c
```

A teljes program hossza 10.000+345+1000 betű (1000 a binárisba)

```
for(;;) {
        sorozat = binárisba(i++);
        if(bonyolultság(sorozat) > 100.000)
}
```

→ sorozat

Kilép és kiírja a sorozatot.

A bonyolultság() függvény implementációja, feltevésünk szerint 10.000 betű

De hát, ez a 11.345 betűs progi is kinyomtatta… Hoppá.

Lásd Javában: 1.8. példa - A Chaitin-Kolmogorov bonyolultság nem kiszámítható!
http://www.tankonyvtar.hu/main.php?objectID=5314387

A program-méret bonyolultság

```
nbatfai@hpserver:~/p1> more nyolcadik.c
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char **argv)
{
  int meddig = atoi(argv[1]);

  for(int i=0; i<meddig; ++i)
    printf("%d", i%2);

  return 0;
}
nbatfai@hpserver:~/p1> wc nyolcadik.c
 13  23 177 nyolcadik.c
```

A program-méret bonyolultság

```
nbatfai@hpserver:~/p1> ./nyolcadik 1
0nbatfai@hpserver:~/p1> ./nyolcadik 10
0101010101nbatfai@hpserver:~/p1> ./nyolcadik 100
01010101010101010101010101010101010101010101010101010101010
01010101010101010101010101010101010101010101010101010101010
```

Sorozat	Sorozat hossza (argv[1])	Program- és input méret (betűkben)	Hányados (méret/hossz)
0	1	177+1	178
0101010101	10	177+2	17.9
$(01)^{50}$	100	177+3	1.80
$(01)^{500}$	1000	177+4	0.181
…	…	…	
…	…	…	

$$(177 + \lg(n) + 1)/n \longrightarrow 0$$

A program-méret bonyolultság

```
nbatfai@hpserver:~/p1> more kilencedik.c
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char **argv)
{
  int meddig = atoi(argv[1]);

  for(int i=0; i<meddig; ++i)
    if(i%2)
      putchar(getchar());
    else
      printf("0");

  return 0;
}
nbatfai@hpserver:~/p1> wc kilencedik.c
 16   25 220 kilencedik.c
```

A program-méret bonyolultság

```
nbatfai@hpserver:~/p1> more 5veletlen.txt
10011
nbatfai@hpserver:~/p1> ./kilencedik 10 <5veletlen.txt >sorozat
nbatfai@hpserver:~/p1> more sorozat
0100000101
```

Sorozat	Sorozat hossza (argv[1])	Program- és input méret (betűkben)	Hányados (méret/hossz)
0	1	220+1	221
0100000101	10	220+2+5	22.7
…	100	220+3+50	2.73
…	1000	220+4+500	0.724
…	…	…	
…	…	…	

$$(177 + \lg(n) + 1 + n/2)/n \longrightarrow 1/2$$

Összefoglalva

Algoritmus mérete/n $\longrightarrow$ 0, azaz algoritmussal generált sorozat nem lehet véletlen

Hatáskör és élettartam C-ben
(érvényességi tartomány)

1) **Belső (lokális, automatikus) változók**
   Hatáskörük a programegységük, amiben definiáltuk őket, élettartamuk, amíg a vezérlés ebben a programegységben van. Inicializálás nincs.

2) **Külső (globális) nevek**
   Hatáskörük a forrás további része (ha korábban, vagy más forrásban is kell, akkor extern deklaráció). Inicializálás van: nulla

Hatáskör és élettartam C-ben

```c
#include <stdio.h>

int
main (void)
{
  int blokkban = 0;

  {
    printf ("%d\n", blokkban);

    {
      int blokkban = 1000;
      printf ("%d\n", blokkban);
      blokkban = blokkban + 1;
      printf ("%d\n", blokkban);
    }

    printf ("%d\n", blokkban);

  }

  printf ("%d\n", blokkban);

  return 0;
}
```

Ebben a blokkban a blokkban változó lokális

Ebben a blokkban ez a név globális

Ebben, a legbelső blokkban ez a név megint lokális

```c
#include <stdio.h>

/* globalis valtozo */
int hivasok_szama = 0;

int
fuggveny (int formalis /* lokalis valtozo, automatikus valtozo */ )
{
/* lokalis valtozo, automatikus valtozo */
  int lokalis = 2;

  printf ("hivasok szama: %d\n", ++hivasok_szama);

  return lokalis * formalis;

}

int
main (void)
{
  /* lokalis valtozo, automatikus valtozo */
  int lokalis, aktualis = 2;

  lokalis = fuggveny (aktualis);

  printf ("%d\n", lokalis);

  lokalis = fuggveny (aktualis);

  printf ("%d\n", lokalis);

  return 0;
}
```

Mi történik, ha a hívasok_szama-nak definícióját leviszed a main elé?

S ha e mellett a régi helyén extern int hivasok_szama; extern deklarációt alkalmazol?

Statikus változók C-ben

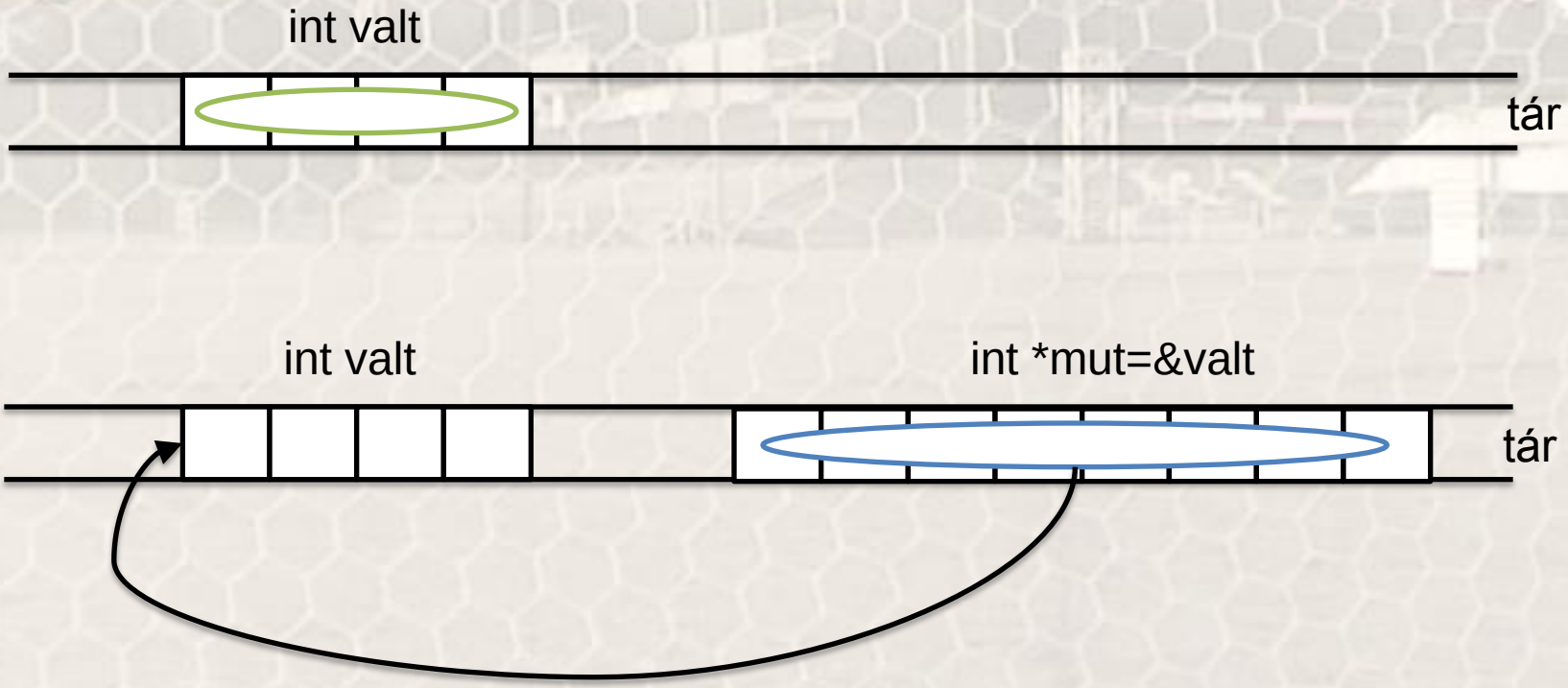
Belső és külső változó is lehet statikus. Inicializálás van: nulla.

```
extern int hivasok_szama;

int
fuggveny (int formalis /* lokalis valtozo, automatikus valtozo */ )
{
/* lokalis valtozo, automatikus valtozo */
  int lokalis = 2;
/* statikus belso */
  static int hivasok_szama2;

  printf ("hivasok szama: %d %d\n", ++hivasok_szama, ++hivasok_szama2);

  return lokalis * formalis;

}

int hivasok_szama = 0;
```

Mi történik, ha a statikus belsőt így definiálod?
static int hivasok_szama2 **= 0**;

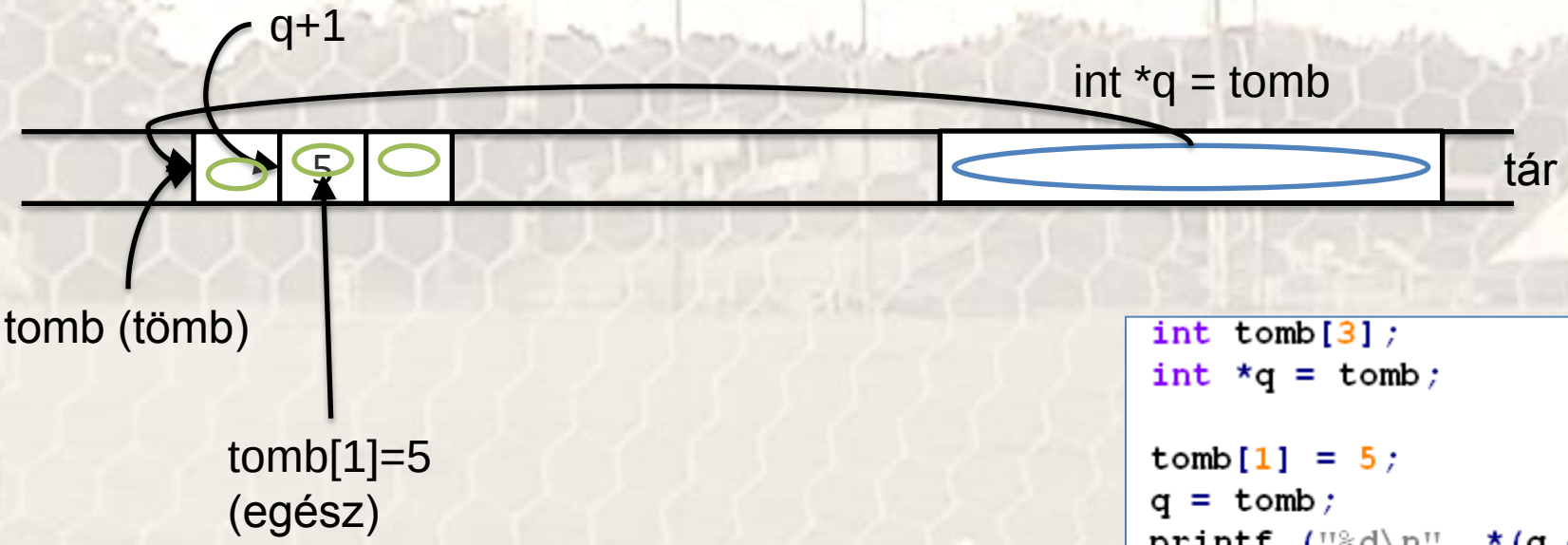
Mutatók (ism.)

```
int valt;          // a valt egy egész
int *mut=&valt;    // a mut egy egészre mutató mutató
```



Mutatók és tömbök

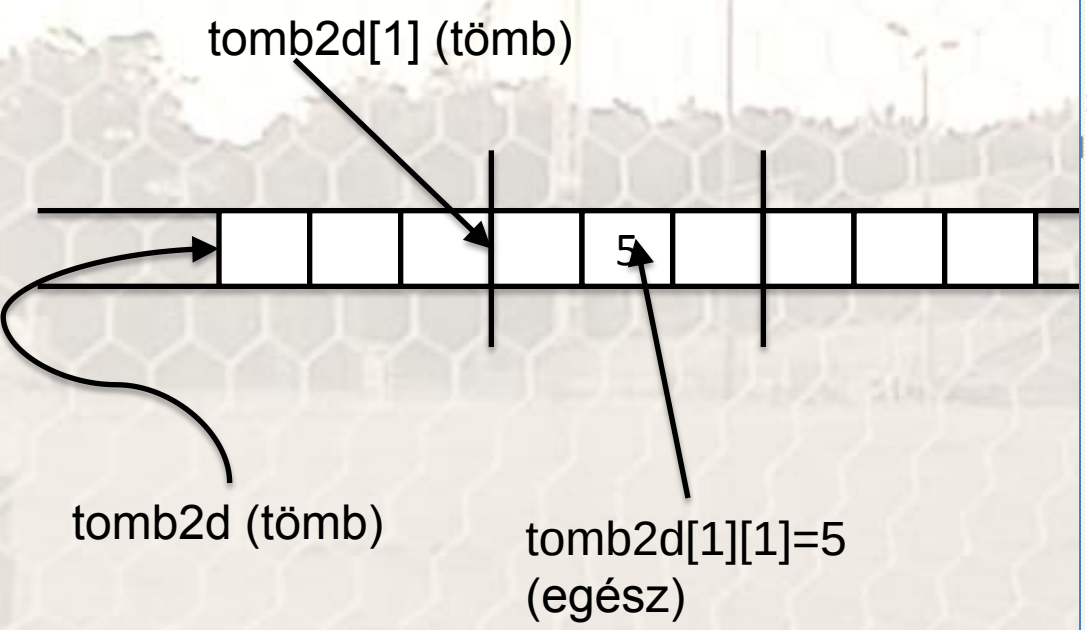
int tomb[3]; // a tomb egy tömb




```c
int tomb[3];
int *q = tomb;

tomb[1] = 5;
q = tomb;
printf ("%d\n", *(q + 1));

*(q + 1) = 6;
printf ("%d\n", tomb[1]);

tomb[1] = 7;
q = &tomb[0];
printf ("%d\n", *(q + 1));
```

Mutatók és tömbök

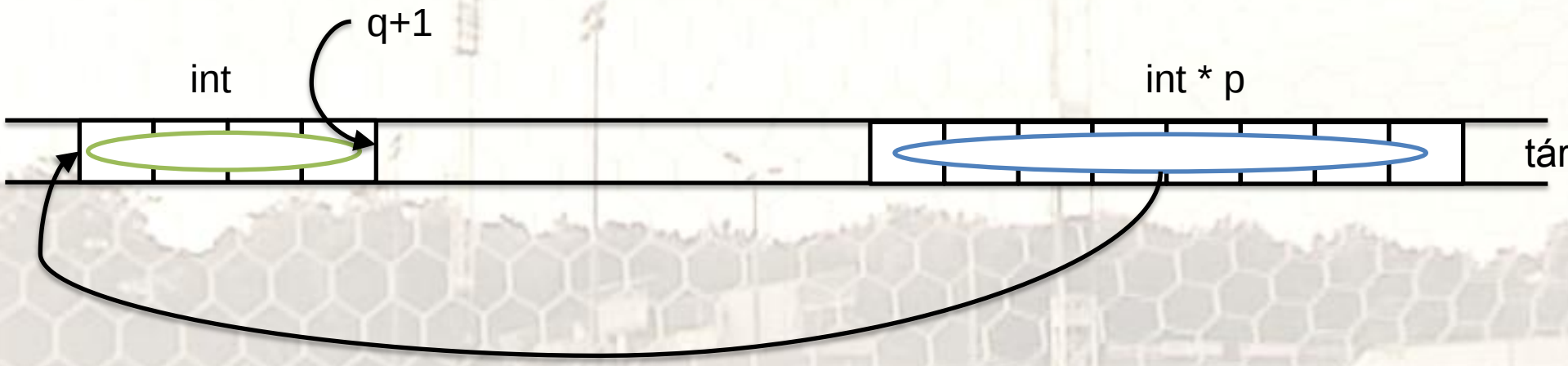
int tomb2d[3][3]; // a tomb2d egy két dimenziós tömb




```c
tomb2d[1][1] = 9;
p = tomb2d[1];
printf ("%d\n", *(p + 1));

tomb2d[1][1] = 10;
p = &tomb2d[1][0];
printf ("%d\n", *(p + 1));
```

```c
#include <stdio.h>

int
main (void)
{
  int tomb2d[3][3];
  int *p;

  tomb2d[1][1] = 5;
  p = (int *)tomb2d;
  printf ("%d\n", *(p + 1 * 3 + 1));

  *(p + 4) = 6;
  printf ("%d\n", tomb2d[1][1]);

  tomb2d[1][1] = 7;
  p = &tomb2d[0][0];
  printf ("%d\n", *(p + 1 * 3 + 1));

  *(p + 4) = 8;
  printf ("%d\n", tomb2d[1][1]);

  return 0;
}
```

Címaritmetika



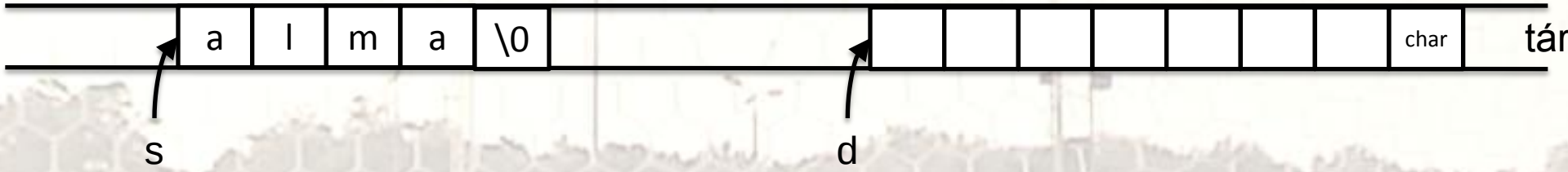
Címaritmetika

```
char *
string_masolo_man_pl_alapjan (char *dest, const char *src, int n)
{
  int i;
  char *p = dest;

/*
  // eredeti
  for (i = 0; i < n && src[i] != '\0'; i++)
    dest[i] = src[i];
*/

/*
  // 1. modositas
  for (i = 0; i < n && src[i]; i++)
    dest[i] = src[i];
*/

  // 2. modositas, char *p felvetele, p[i]='\0' es return p
  for (i = 0; i < n && (*dest++ = *src++); i++)
    ;

  for (; i < n; i++)
    p[i] = '\0';

//  return dest;
  return p;
}
```

*d++ = * s++

Címaritmetika

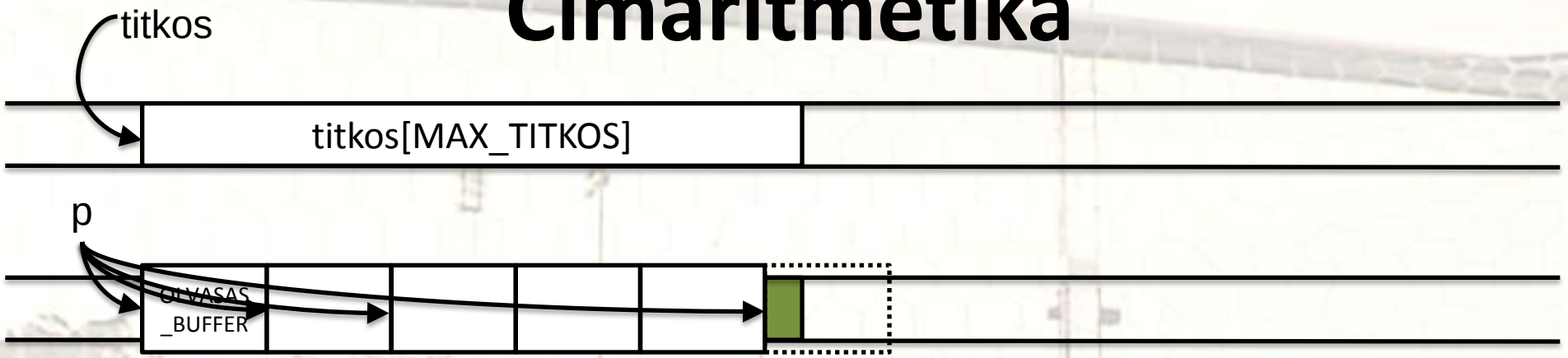
*d++ = * s++



*d++ = * s++ (először feldolgoz, utána inkrementál)

*d++ = * s++

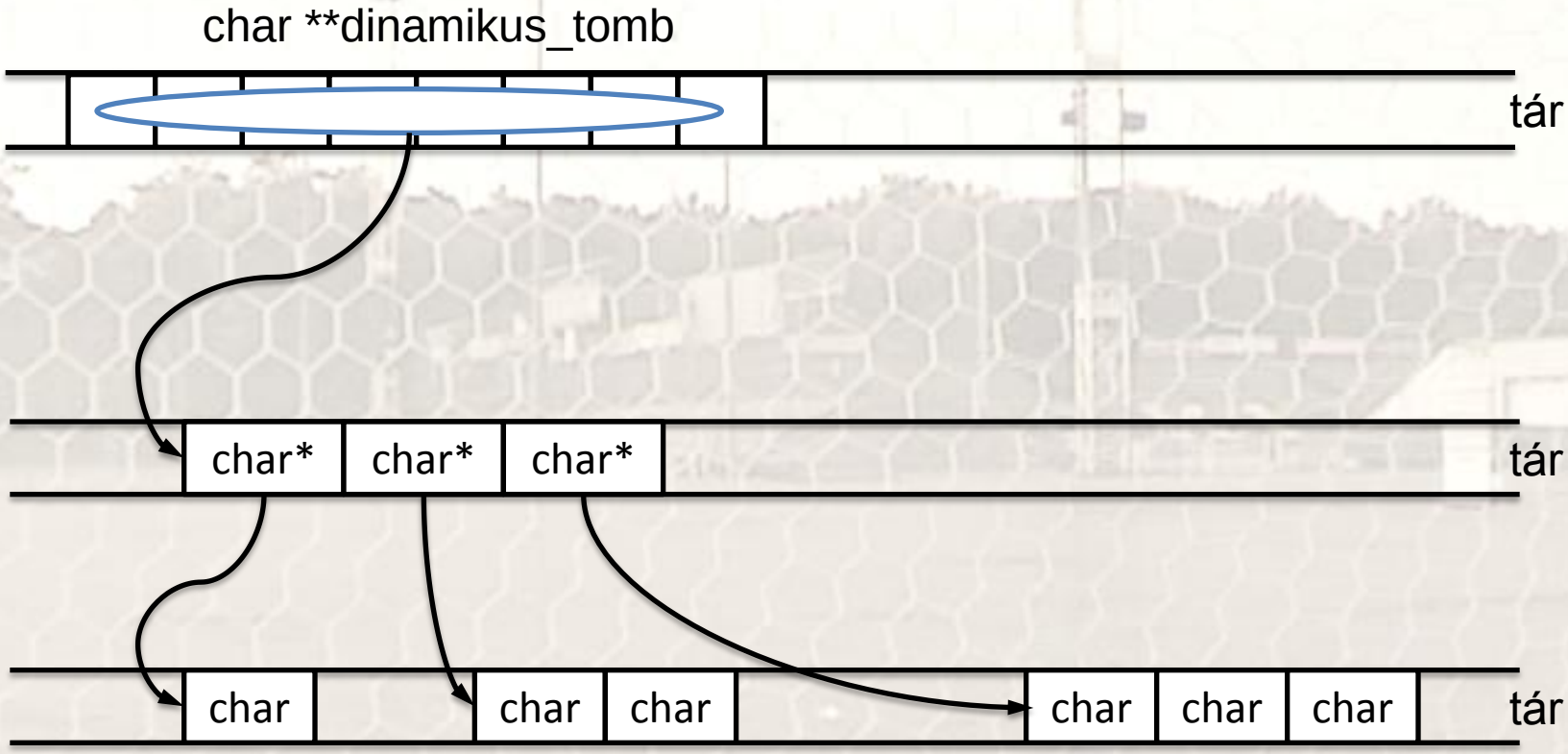
Címaritmetika



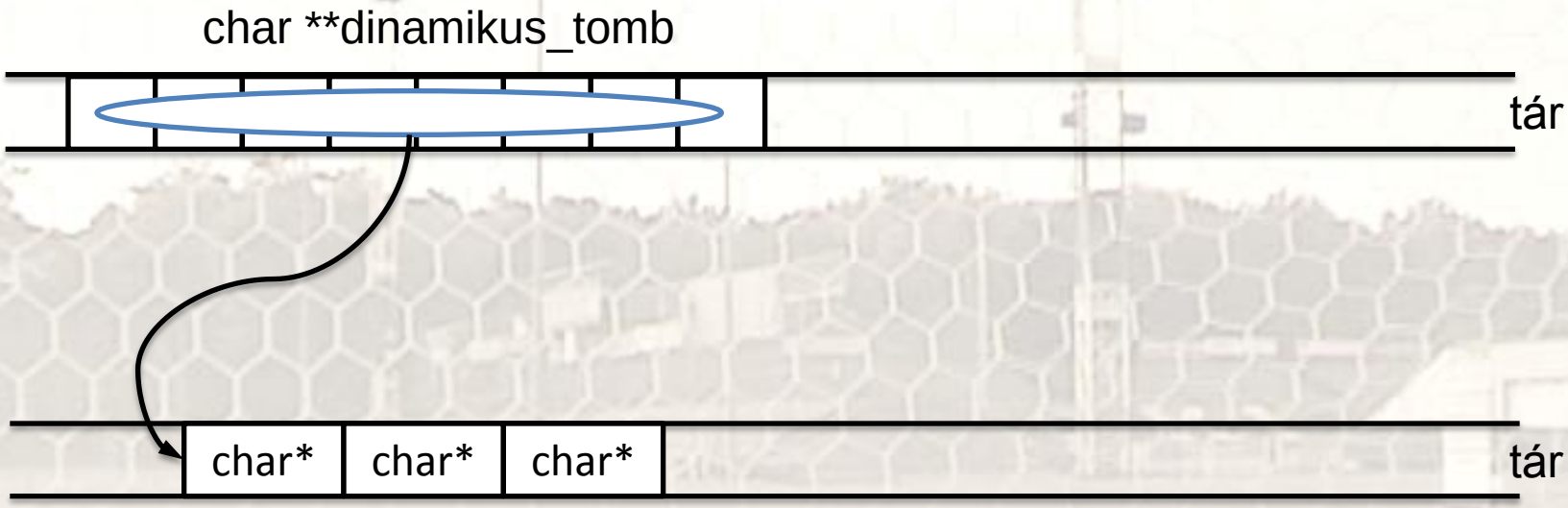

```
int
main (void)
{

  char kulcs[KULCS_MERET];
  char titkos[MAX_TITKOS];
  char *p = titkos;
  int olvasott_bajtok;

  // titkos fajt pufferelt berantasa
  while ((olvasott_bajtok =
          read (0, (void *) p,
                (p - titkos + OLVASAS_BUFFER <
                 MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - p)))
    p += olvasott_bajtok;

  // maradek hely nullazasa a titkos bufferben
  for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
    titkos[p - titkos + i] = '\0';
```

http://progpater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia

Mutatók és többdimenziós tömbök



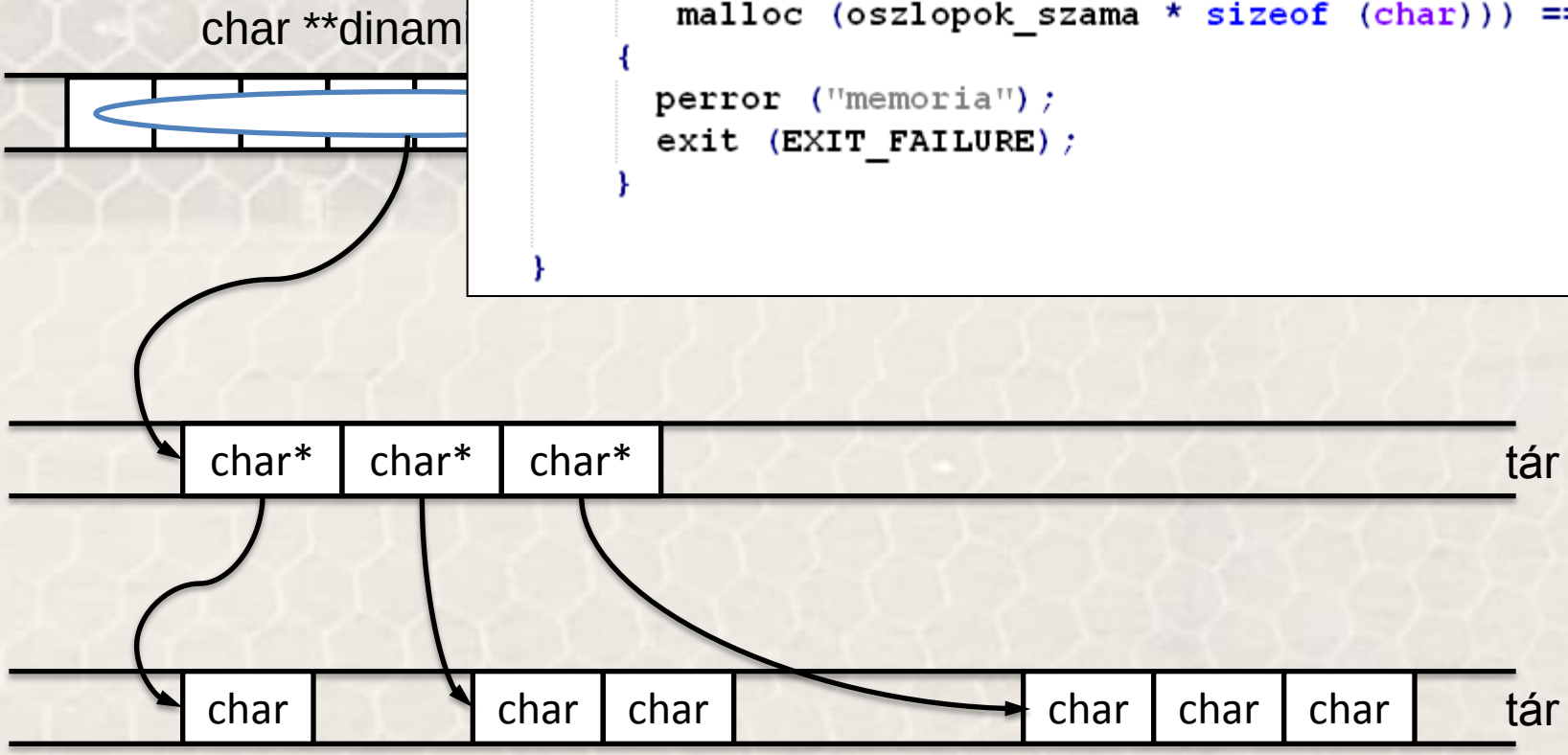
Mutatók és többdimenziós tömbök



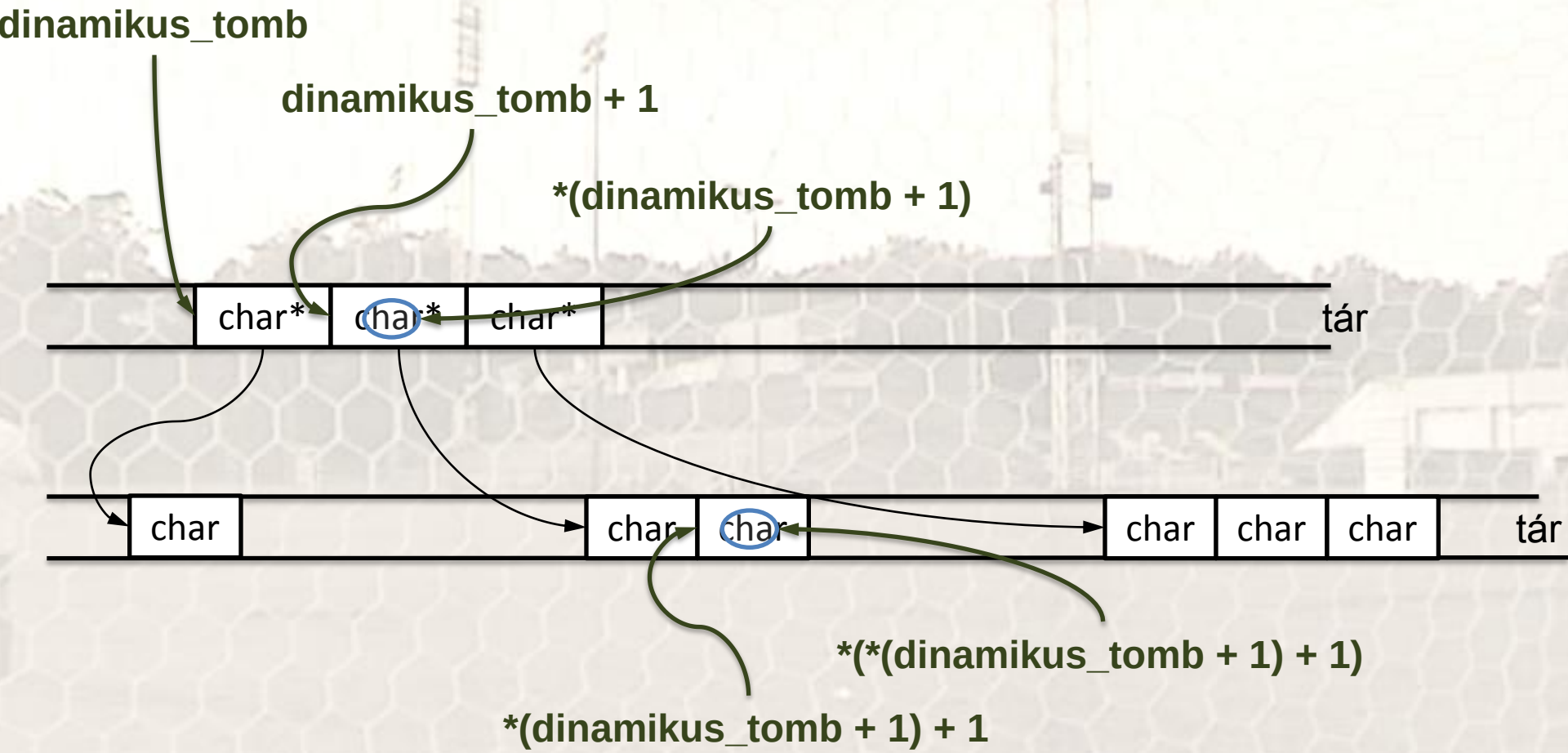

```
if ((dinamikus_tomb = (char **)
     malloc (sorok_szama * sizeof (char *))) == NULL)
  {
    perror ("memoria");
    exit (EXIT_FAILURE);
  }
```

Mutatók és többdimenziós tömbök

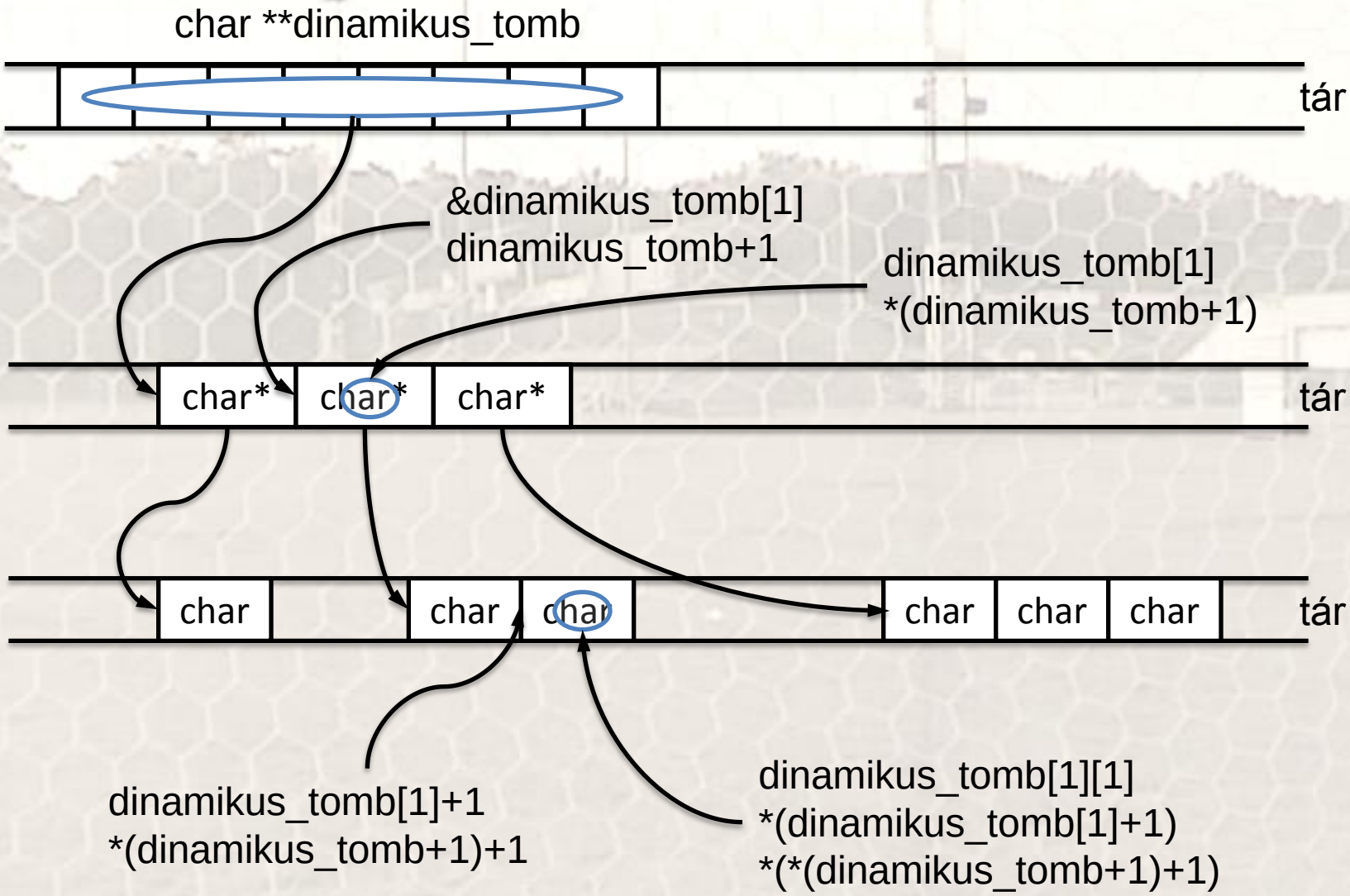
```
for (int i = 0; i < sorok_szama; ++i)
  {
    oszlopok_szama = i + 1;

    if ((dinamikus_tomb[i] = (char *)
         malloc (oszlopok_szama * sizeof (char))) == NULL)
      {
        perror ("memoria");
        exit (EXIT_FAILURE);
      }

  }
```



((dinamikus_tomb + 1) + 1) = ‘y’;

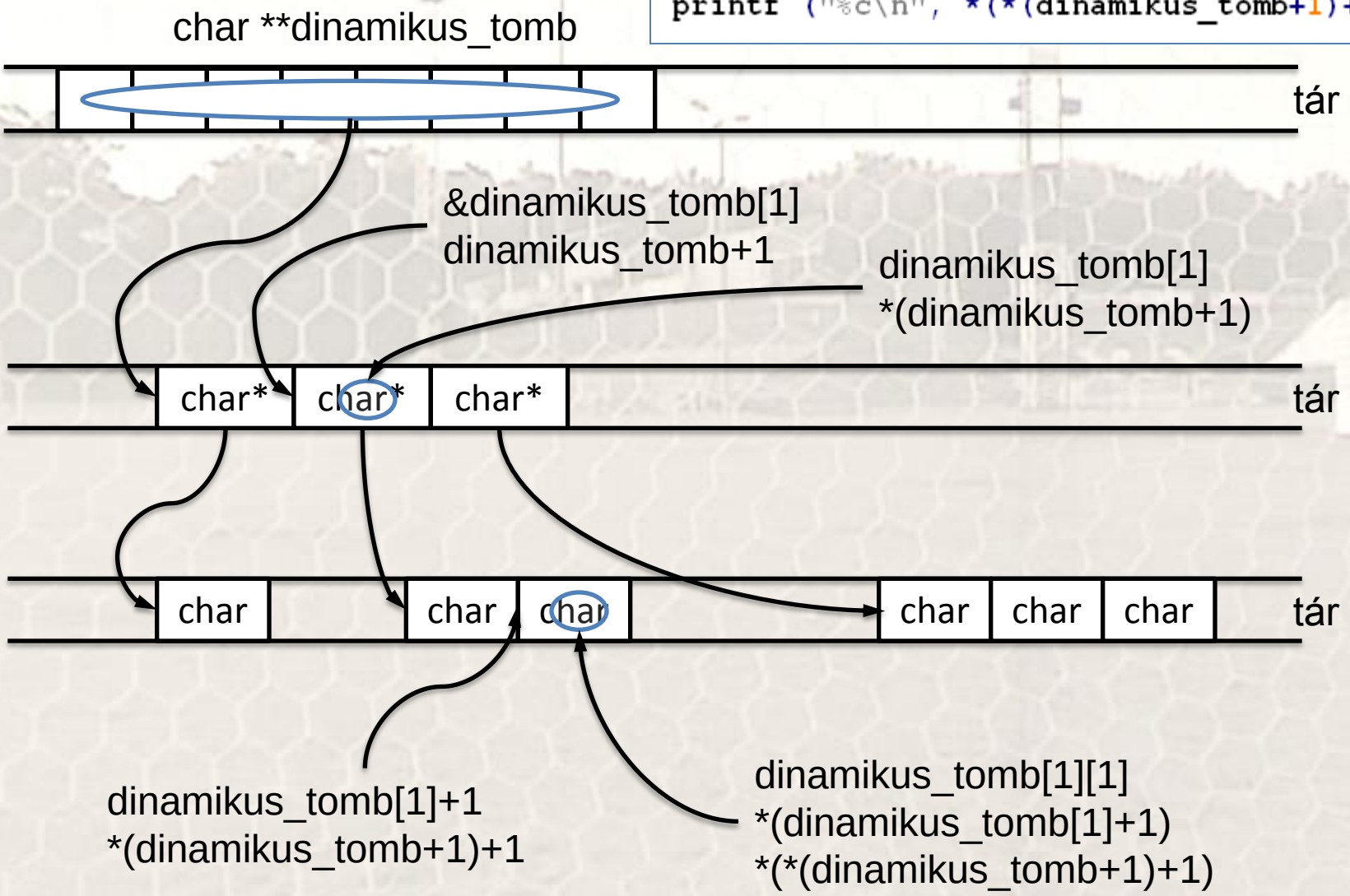


Mutatók és többdimenziós tömbök

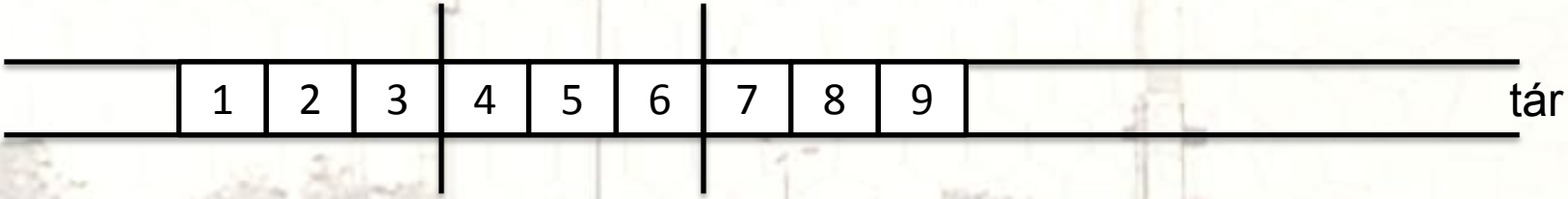


Mutatók és többdimenziós tön

```
dinamikus_tomb[1][1] = 'x';
printf ("%c\n", *(*(dinamikus_tomb + 1) + 1));

*(*(dinamikus_tomb + 1) + 1) = 'y';
printf ("%c\n", dinamikus_tomb[1][1]);
printf ("%c\n", *(dinamikus_tomb[1]+1));
printf ("%c\n", *(*(dinamikus_tomb+1)+1));
```



A két dimenziós tömb (ism.) és a „megfelelő” mutató tömb



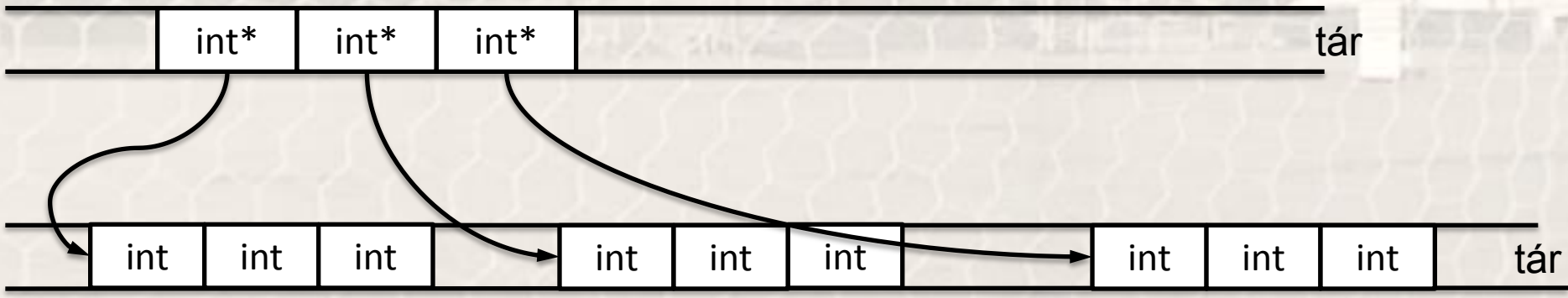

```c
#include <stdio.h>

int
main (void)
{
  int i, j, tomb2d[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
  };

  for (i = 0; i < 3; ++i)
    {
      for (j = 0; j < 3; ++j)
        printf ("[%d][%d]=%d ", i, j, tomb2d[i][j]);

      printf ("\n");
    }
  return 0;
}
```

Helyigény: 9 db int

A két dimenziós tömb és a „megfelelő” mutató tömb

Helyigény: 9 db int, +3 db int*

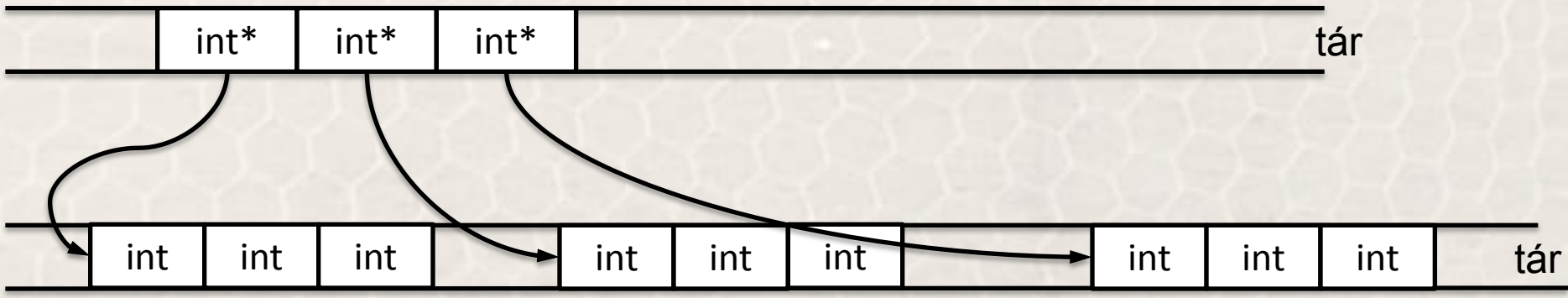
int *dinamikus_tomb[3];



A két dimenziós tömb és a „megfelelő” mutató tömb

```
int sorok_szama = 3;
int oszlopok_szama = 3;
int *dinamikus_tomb[sorok_szama];

for (int i = 0; i < sorok_szama; ++i)
  if ((dinamikus_tomb[i] = (int *)
       malloc (oszlopok_szama * sizeof (int))) == NULL)
    {
      perror ("memoria");
      exit (EXIT_FAILURE);
    }
```

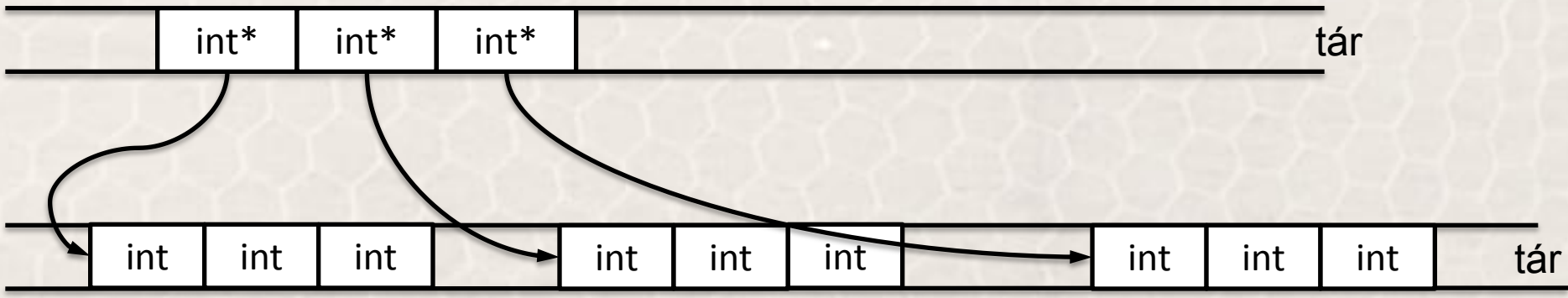
int *dinamikus_tomb[3];



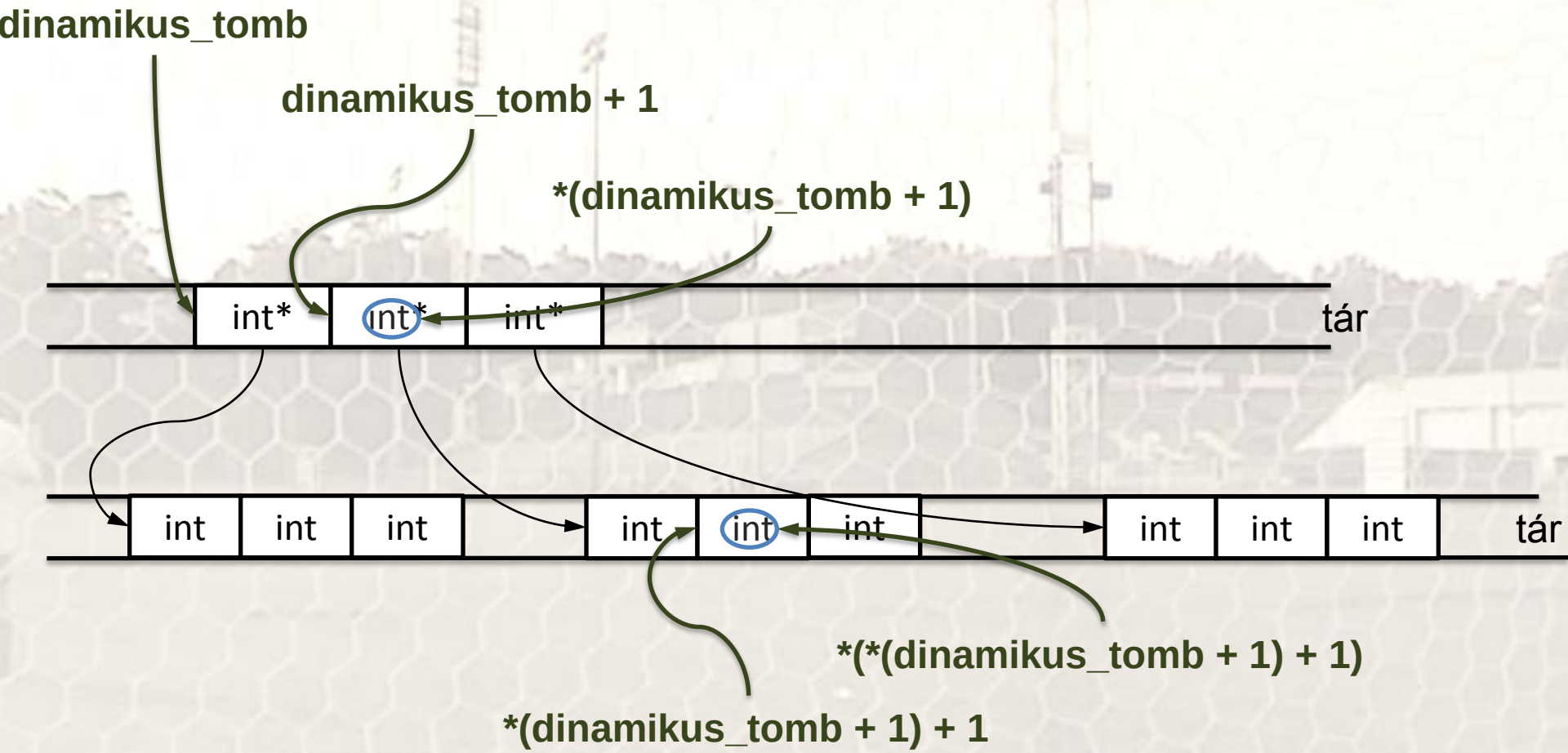
Mutató tömb

```
dinamikus_tomb[1][1] = 5;
printf ("%d\n", *(*(dinamikus_tomb + 1) + 1));
printf ("%d\n", *(*(&dinamikus_tomb[1]) + 1));

*(*(dinamikus_tomb + 1) + 1) = 6;
printf ("%d\n", dinamikus_tomb[1][1]);
printf ("%d\n", *(dinamikus_tomb[1] + 1));
printf ("%d\n", *(*(dinamikus_tomb + 1) + 1));

*(dinamikus_tomb[1] + 1) = 7;
printf ("%d\n", dinamikus_tomb[1][1]);
printf ("%d\n", *(dinamikus_tomb[1] + 1));
printf ("%d\n", *(*(dinamikus_tomb + 1) + 1));
```

int *dinamikus_tomb[3];



((dinamikus_tomb + 1) + 1) = 6;

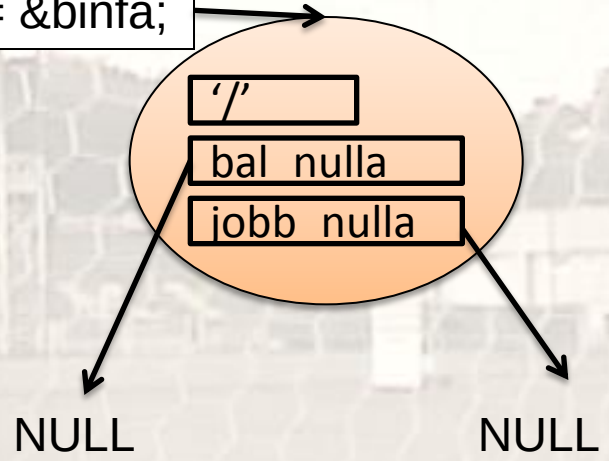


Dinamikus tárkezelés

```
typedef struct binfa
{
  int ertek;
  struct binfa *bal_nulla;
  struct binfa *jobb_egy;

} BINFA, *BINFA_PTR;

BINFA_PTR
uj_elem ()
{
  BINFA_PTR p;

  if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
    {
      perror ("memoria");
      exit (EXIT_FAILURE);
    }
  return p;
}
```

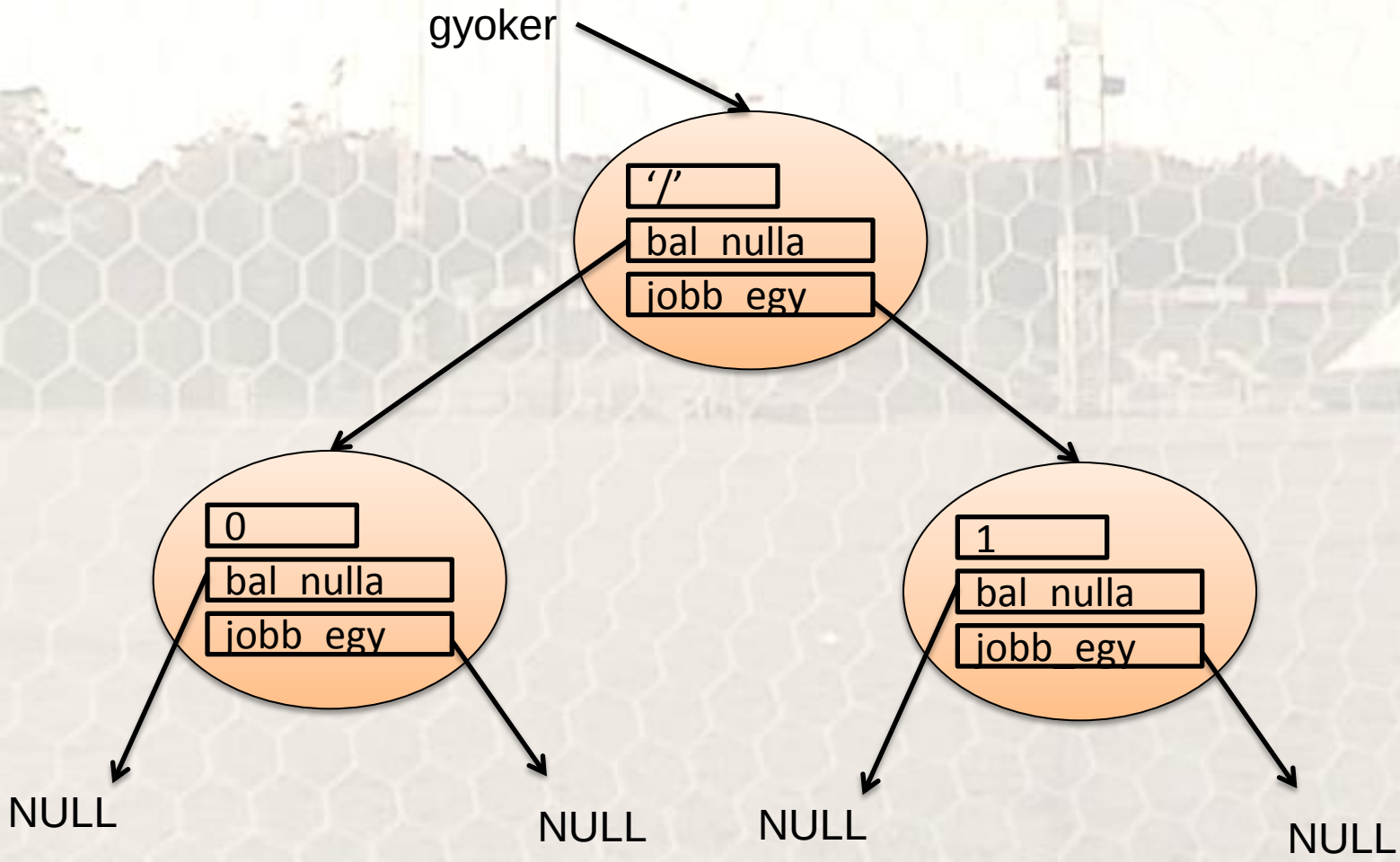
BINFA binfa = {'/', NULL, NULL};

BINFA_PTR binfap = &binfa;



http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

Dinamikus tárkezelés



http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

Labor

Az első néhány laboron mindenképpen egyszerű szövegszerkesztőt és parancssort használjunk! A PP javasolta manuál lapokat mindig nyissuk ki, nézzük meg!

PageRank (1)

Alapcikk

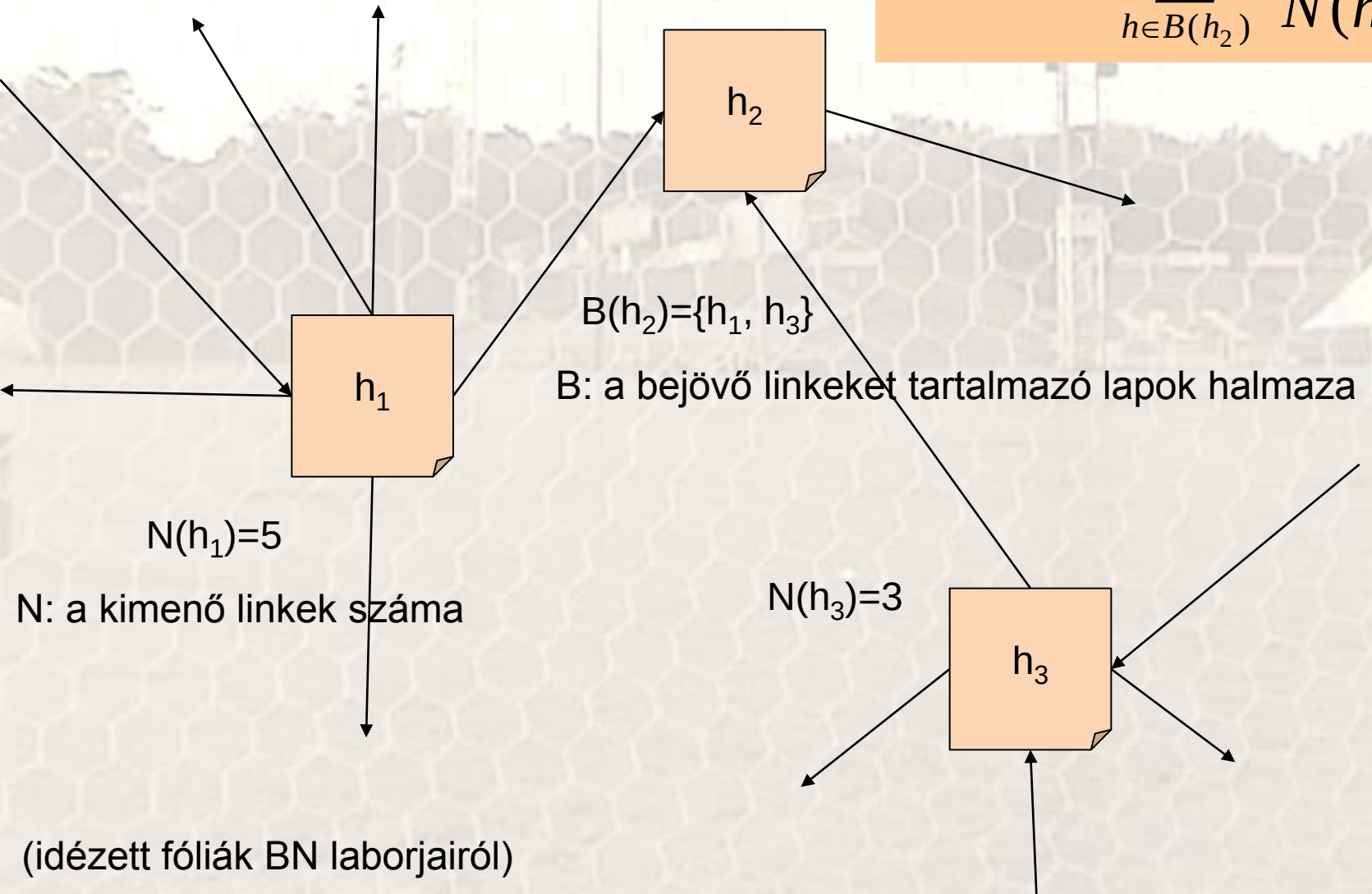
Page, Lawrence; Brin, Sergey; Motwani, Rajeev; Winograd, Terry.
The PageRank Citation Ranking: Bringing Order to the Web.
http://dbpubs.stanford.edu:8090/pub/1999-66/

Ötlet

Azok a weblapok jobb minőségűek, amelyekre jobb minőségű lapok mutatnak.

PageRank (2)

$$PR(h_2) = \sum_{h \in B(h_2)} \frac{PR(h)}{N(h)} \quad (1)$$



$B(h_2)=\{h_1, h_3\}$

B: a bejövő linkeket tartalmazó lapok halmaza

$N(h_1)=5$

N: a kimenő linkek száma

$N(h_3)=3$

(idézett fóliák BN laborjairól)

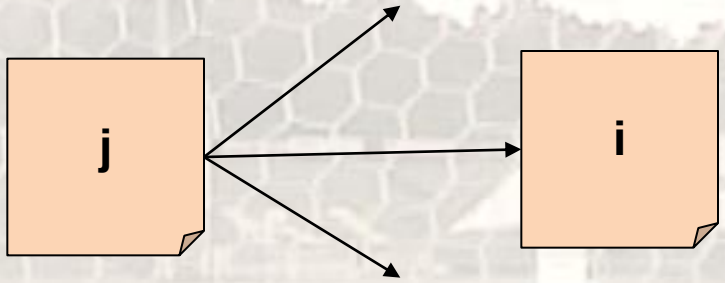
PageRank (4)

Írjuk fel (1)-et tömörebben: a linkmátrix

$$L = (l_{i,j})$$

$$l_{i,j} = \begin{cases} \dfrac{1}{N(j)}, & j \longrightarrow i \\ 0 \end{cases}$$

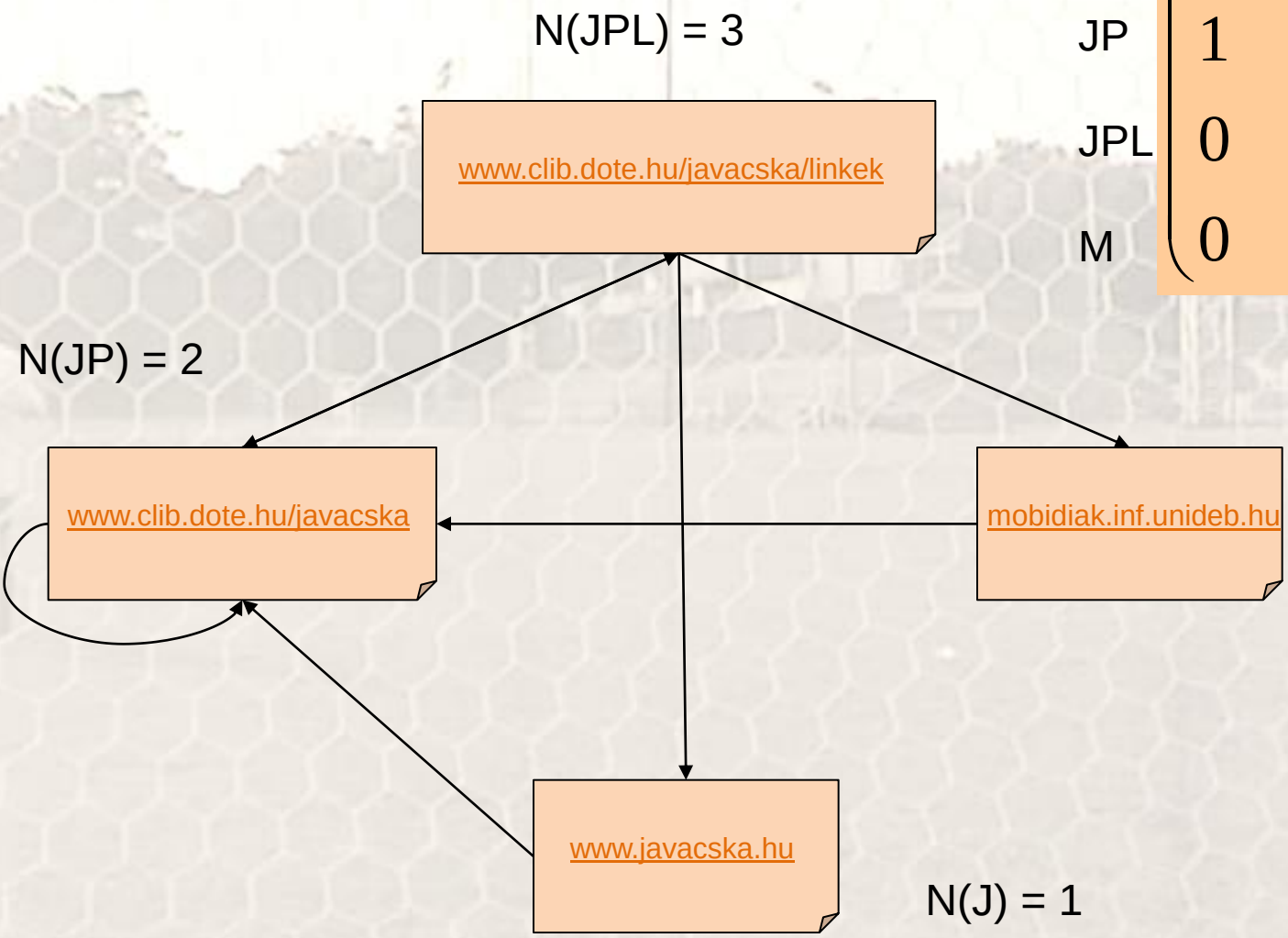
$$L = \begin{pmatrix} l_{1,1} & \cdots & l_{1,j} & \cdots & l_{1,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ l_{i,1} & \cdots & l_{i,j} & \cdots & l_{i,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ l_{n,1} & \cdots & l_{n,j} & \cdots & l_{n,n} \end{pmatrix}$$



PageRank (5)

Példa a linkmátrixra

	J	JP	JPL	M
J	0	0	1/3	0
JP	1	1/2	1/3	1
JPL	0	1/2	0	0
M	0	0	1/3	0



PageRank (6)

Írjuk fel (1)-et a linkmátrixal

$$\begin{pmatrix} PR(h_1) \\ \vdots \\ PR(h_n) \end{pmatrix}$$

$$\begin{pmatrix} l_{1,1} & \cdots & l_{1,j} & \cdots & l_{1,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ l_{i,1} & \cdots & l_{i,j} & \cdots & l_{i,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ l_{n,1} & \cdots & l_{n,j} & \cdots & l_{n,n} \end{pmatrix}$$

$$\begin{pmatrix} PR(h_1) = \sum_{j=1}^{n} l_{1,j} PR(h_j) \\ \vdots \\ PR(h_i) = \sum_{j=1}^{n} l_{i,j} PR(h_j) \\ \vdots \\ PR(h_n) = \sum_{j=1}^{n} l_{n,j} PR(h_j) \end{pmatrix}$$

Tehát ha h jelöli a PR vektort, akkor h=Lh

Linalg kedvelőknek: a PageRank vektor az L linkmátrix 1 sajátértékhez tartozó sajátvektora.

PageRank (7)

Laborfeladat

- Billentyűzzünk be gyorsan egy olyan C progit, ami kiszámítja a PageRank vektort a mutatott példához! Addig billentyűzzünk most, amíg ez nem jelenik meg a képernyőn:
  PR(J)=.09, PR(JP)=.54, PR(JPL)=.27, PR(M)=.09

A következő poszt mutatta forráscsipetekből induljunk ki:
http://progpater.blog.hu/2011/02/13/bearazzuk_a_masodik_labort

Beár

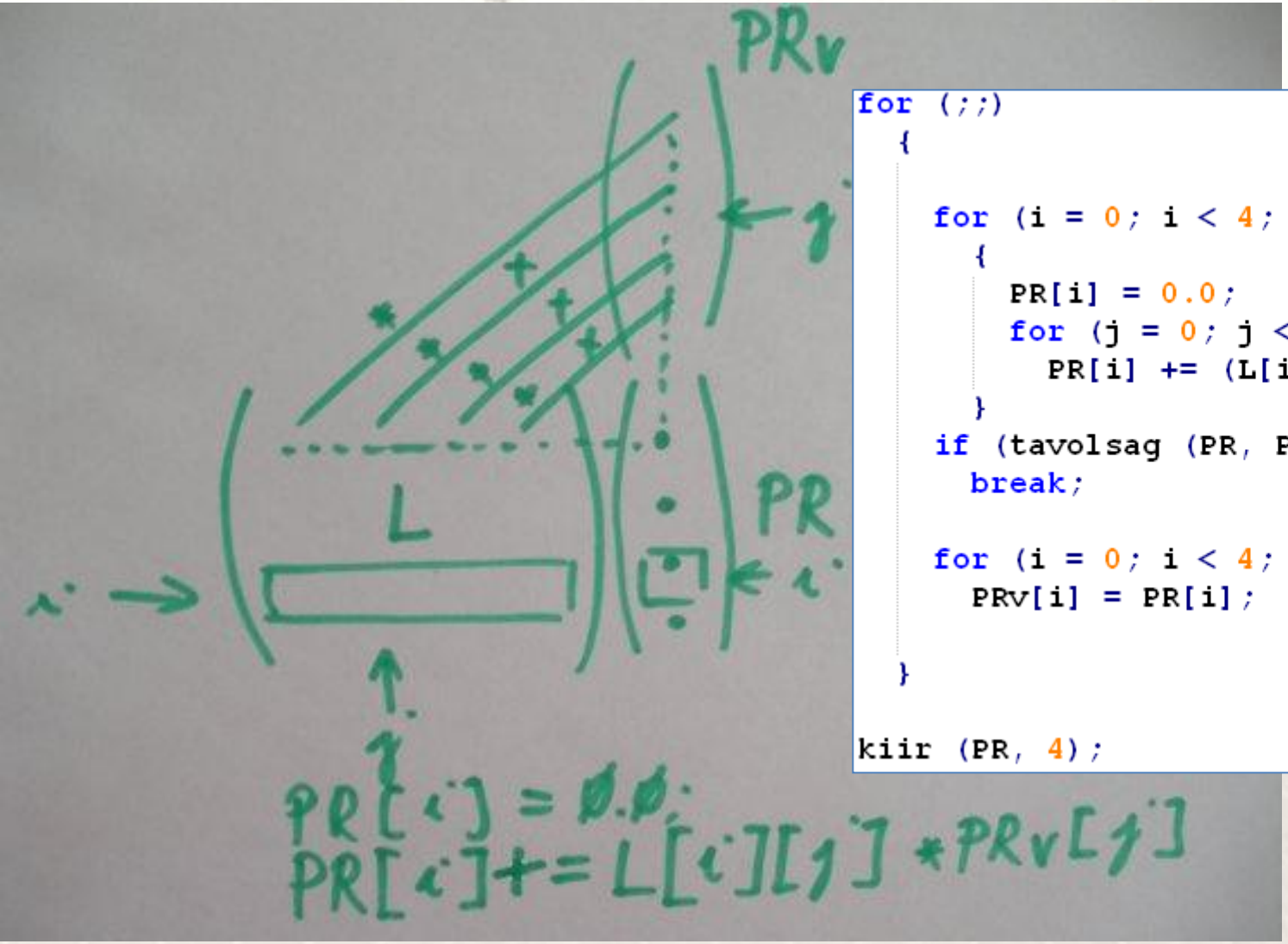
http://p

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

void
kiir (double tomb
{
/* Irasd ki a par
}

double
tavolsag (double
{

// nekem itt volt

  for (i = 0; i <
    osszeg += (PR

  return sqrt (os
}
```

```c
int
main (void)
{

  double L[4][4] = {
    {0.0, 0.0, 1.0 / 3.0, 0.0},
    {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
    {0.0, 1.0 / 2.0, 0.0, 0.0},
    {0.0, 0.0, 1.0 / 3.0, 0.0}
  };

  double PR[4] = { 0.0, 0.0, 0.0, 0.0 };
  double PRv[4] = { 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0 };

  int i, j;

  for (;;)
    {

    // nekem itt volt kód...

      if (tavolsag (PR, PRv, 4) < 0.0000000001)
        break;

    // nekem itt volt kód...

    }

  kiir (PR, 4);

  return 0;
}
```

Beárazzuk a második labort

http://progpater.blog.hu/2011/02/13/bearazzuk_a_masodik_labort




```
for (;;)
  {

    for (i = 0; i < 4; ++i)
      {
        PR[i] = 0.0;
        for (j = 0; j < 4; ++j)
          PR[i] += (L[i][j] * PRv[j]);
      }
    if (tavolsag (PR, PRv, 4) < 0.00000001
      break;

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

  }

kiir (PR, 4);
```

PageRank (7)

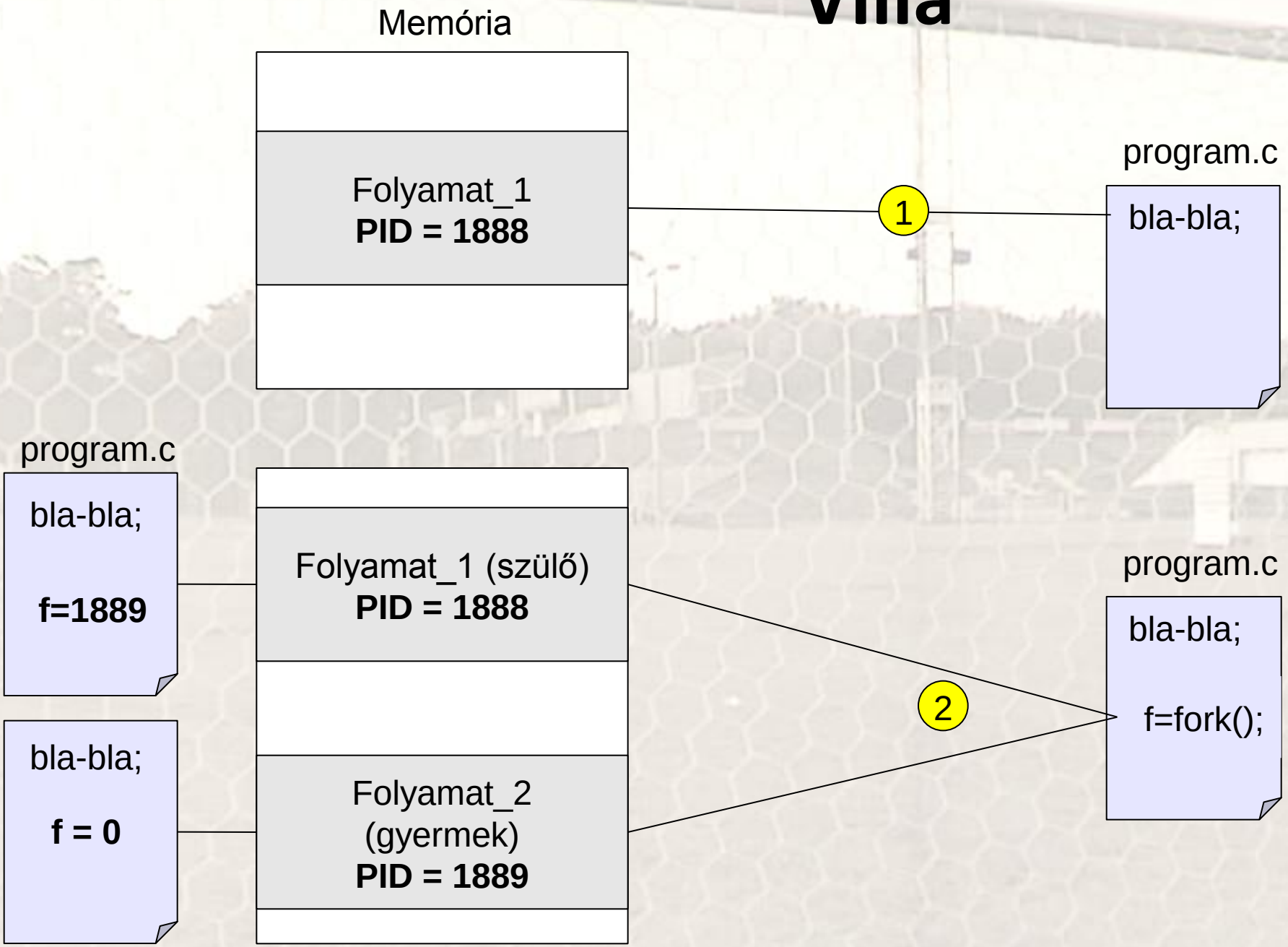
További esetleges feladatok

- Mi történik, ha egy lapnak nincs kimenő linkje? (Az előző program felhasználásával válaszoljunk!) Töröljük például a JP ← M linket!
- Mi történik, ha egy lap csak magára mutat? (Az előző program felhasználásával válaszoljunk!) JP → M linket módosítsuk M → M linkre.
- Mi történik, ha az alábbi módon számítjuk ki a PR-t? (Az előző program felhasználásával válaszoljunk!)

$$PR(h_i) = d \sum_{j=1}^{n} l_{i,j} PR(h_j) + (1-d)/n$$

Felhasznált és további irodalom:
http://www.cs.duke.edu/~junyang/courses/cps296.1-2002-spring/lectures/02-web-search.

Villa


Memória
Folyamat_1
PID = 1888
program.c
bla-bla;
1
program.c
bla-bla;
f=1889
bla-bla;
f = 0
Folyamat_1 (szülő)
PID = 1888
Folyamat_2
(gyermek)
PID = 1889
2
program.c
bla-bla;
f=fork();

VILLA készítése

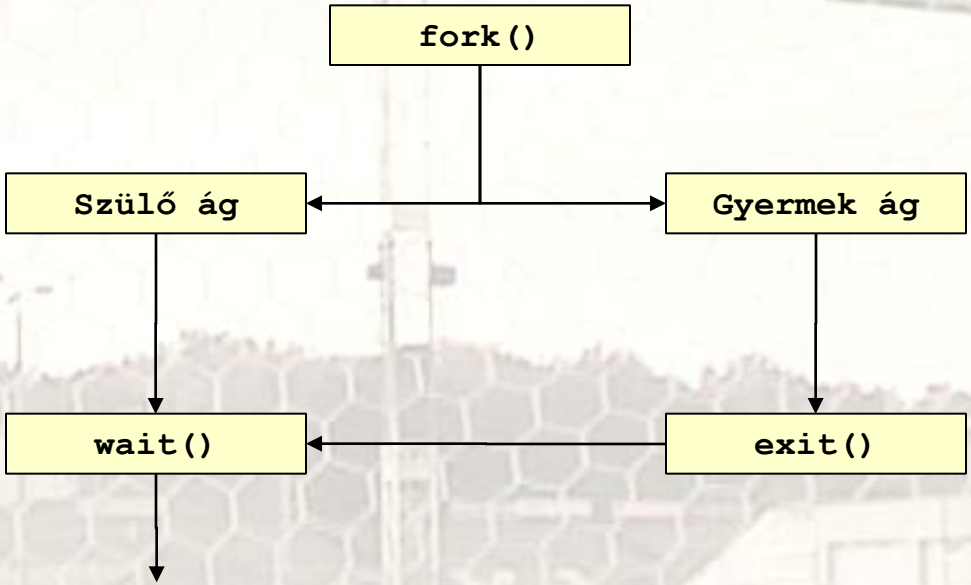
```
...

  if((gyermekem_pid = fork()) == 0)
  {
       // ITT FOLYTATÓDIK A VEZÉRLÉS A GYERMEKBEN, mert itt gyermekem_pid = 0
  }
  else if(gyermekem_pid > 0)
  {
       // ITT FOLYTATÓDIK A VEZÉRLÉS A SZÜLŐBEN, mert itt gyermekem_pid = a
       //                                        gyermek PID-jével
  }
  else
  {
       // Hiba ág: ide kerül a vezérlés, ha nem sikerült a forkolás.
  }
...
```

```
$ man fork

FORK(2)                    Linux Programmer's Manual                    FORK(2)

...

RETURN VALUE
       On success, the PID of the child process is returned in the parent's
       thread of execution, and a 0 is returned in the child's thread of exe-
       cution.  On failure, a -1 will be returned in the parent's context, no
       child process will be created, and errno will be set appropriately.
...
```

A fork() tipikus használata

PP 40

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

int
main(void)
{
  int gyermekem_pid;
  int statusz;
  if((gyermekem_pid = fork()) == 0)
  {
    char *args[] = {"/bin/ls", NULL};
    execve("/bin/ls", args, NULL);
    exit(0);
  }
  else if(gyermekem_pid > 0)
  {
    wait(&statusz);
  }
  else
  {
   exit(-1);
  }
  return 0;
}
```

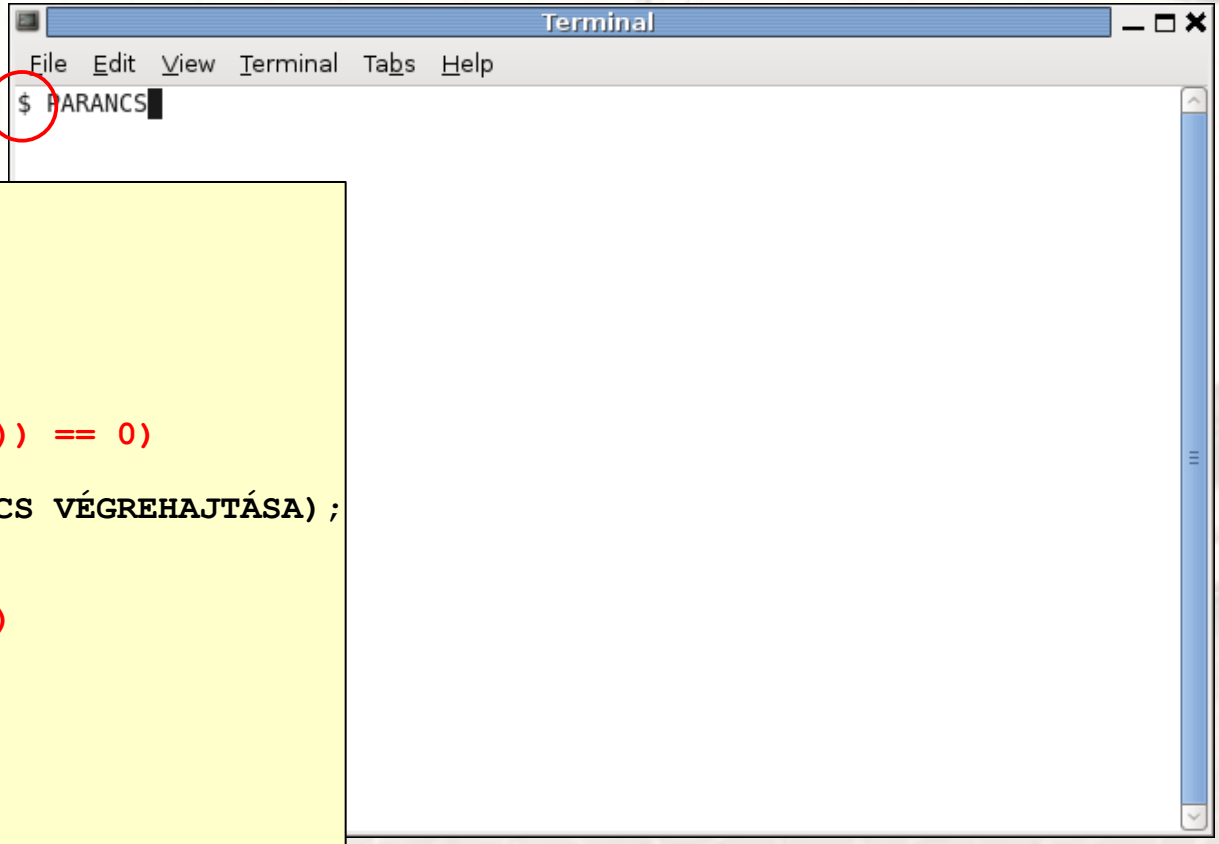


A gyermek folyamat elvégzi az ls parancs futtatását, majd kilép (befejeződik)

Közben a szülő folyamat várakozik a gyermek befejeződésére

Miért jó ez?

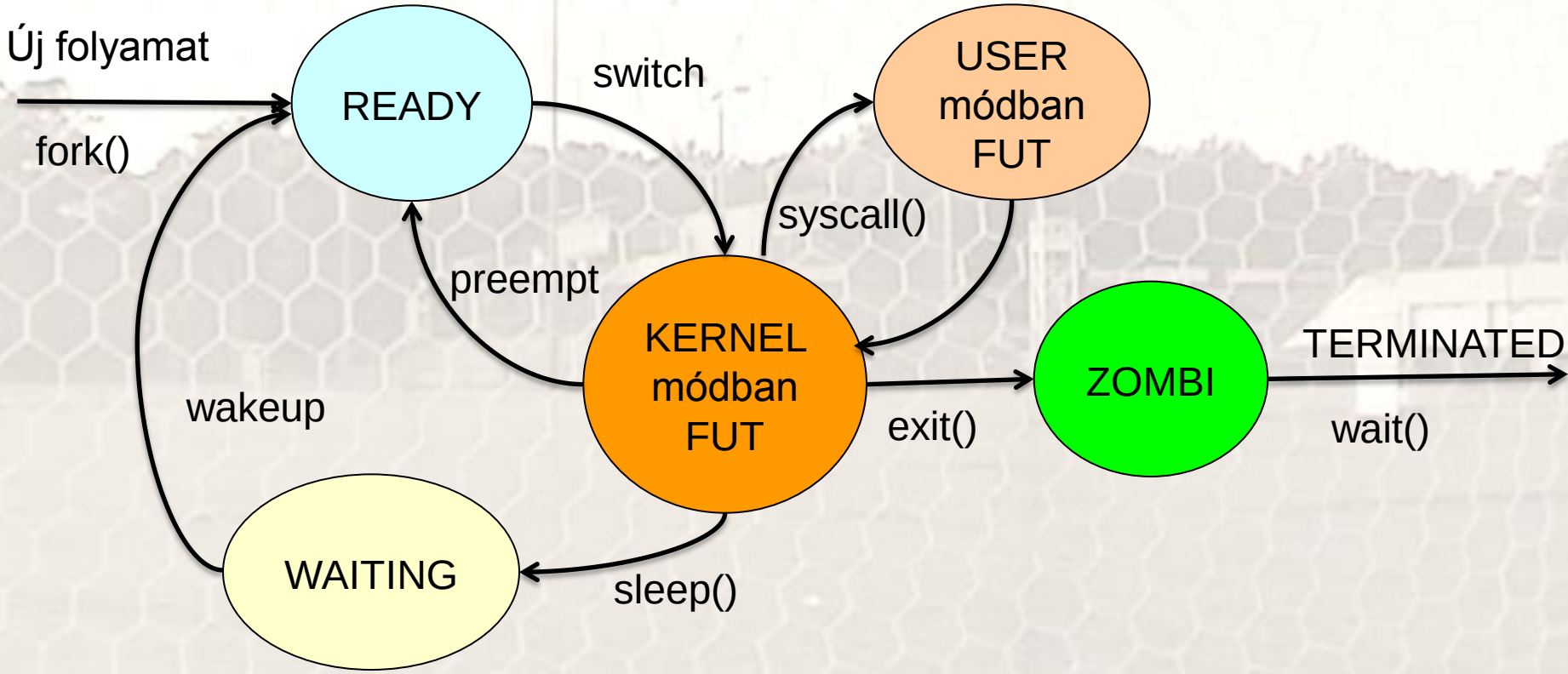
A parancssor




```
...
for(;;)
{
  prompt_kiírása("$");
  parancssor_beolvasása();

  if((gyermekem_pid = fork()) == 0)
  {
    execve(BEOLVASOTT PARANCS VÉGREHAJTÁSA);
    exit(0);
  }
  else if(gyermekem_pid > 0)
  {
    wait(&statusz);
  }
  else
  {
   exit(-1);
  }
}
...
```

A parancsértelmező (shell) szervezése

A UNIX processz: árvák és zombik



http://progpater.blog.hu/2011/02/07/zombiveszely_1

Laborkártyák

Mit jelent az, hogy C-ben a paraméterátadás érték szerinti, sorrendi kötés van és nincs szám szerinti egyeztetés?

Laborkártyák

Állítsd precedencia szerint sorrendbe! Mondanom sem kell: könyv, jegyzet, bármi használható közben, mint a vizsgán, de itt gyorsan jöjjenek a válaszok! Természetesen az operátorok jelentését is tudni kell.

>=	*
+=	*=
->	.
-=	&
<<	&&

Mi a különbség az if-ek fejében a két kifejezés között (ha van)

if(b & 0x80 == 0) …

if((b & 0x80) == 0) …

Laborkártyák

Alábbi kód kapcsán: mit mond a C a + operátor operandusainak kiértékelési sorrendjéről?

```c
#include <stdio.h>

int
novel_eggyel (int *a)
{
  return ++ * a;
}

int
csokkent_eggyel (int *a)
{
  return -- * a;
}


int
main (void)
{
  int a = 0;
  printf ("%d\n", novel_eggyel (&a) + csokkent_eggyel (&a));

  a = 0;
  printf ("%d\n", csokkent_eggyel (&a) + novel_eggyel (&a));

  return 0;
}
```

1
-1

Laborkártyák

```c
#include <stdio.h>

int
main (void)
{

  int blokkban = 0;

  {
    int blokkban = 100;
    printf ("%d\n", blokkban);

    {
      int blokkban = 10000;
      printf ("%d\n", blokkban);
      blokkban = blokkban + 1;
      printf ("%d\n", blokkban);
    }

    printf ("%d\n", blokkban);

  }

  printf ("%d\n", blokkban);

  return 0;
}
```

Mit ír ki ez a program?

Laborkártyák

```c
#include <stdio.h>

int
main (void)
{

  int blokkban = 0;

  {
    printf ("%d\n", blokkban);

    {
      int blokkban = 1000;
      printf ("%d\n", blokkban);
      blokkban = blokkban + 1;
      printf ("%d\n", blokkban);
    }

    printf ("%d\n", blokkban);

  }

  printf ("%d\n", blokkban);

  return 0;
}
```

Mit ír ki ez a program?

```
char *
string_masolo_man_pl_alapjan (char *dest, const char *src, int n)
{
  int i;
  char *p = dest;

/*
  // eredeti
  for (i = 0; i < n && src[i] != '\0'; i++)
    dest[i] = src[i];
*/

/*
  // 1. modositas
  for (i = 0; i < n && src[i]; i++)
    dest[i] = src[i];
*/

  // 2. modositas, char *p felvetele, p[i]='\0' es return p
  for (i = 0; i < n && (*dest++ = *src++); i++)
    ;

  for (; i < n; i++)
    p[i] = '\0';

//  return dest;
  return p;
}
```

Mi a működésbeli különbség az „eredeti”, az „1. módosítás” és a „2. módosítás” között?

http://progpater.blog.hu/2011/02/06/a_forras_szovete_avagy_dest_src

```c
#include <stdio.h>

extern int hivasok_szama;

int
fuggveny (int formalis /* lokalis valtozo, automatikus valtozo */ )
{
/* lokalis valtozo, automatikus valtozo */
  int lokalis = 2;
/* statikus belso */
  static int hivasok_szama2;

  printf ("hivasok szama: %d %d\n", ++hivasok_szama, ++hivasok_szama2);

  return lokalis * formalis;

}

int hivasok_szama = 0;

int
main (void)
{
  /* lokalis valtozo, automatikus valtozo */
  int lokalis, aktualis = 2;

  lokalis = fuggveny (aktualis);

  aktualis = fuggveny (aktualis);

  return 0;
}
```

Mit ír ki ez a program?

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
int kulso = 0;
static int statikus_kulso = 0;
void
valtozok (void)
{
  int belso = 0;
  static int statikus_belso = 0;
  int gyermekem_pid;
  int statusz;
  if ((gyermekem_pid = fork ()) == 0)
    {
      printf ("GY: %d %d %d %d\n", ++kulso, ++statikus_kulso,
              ++belso, ++statikus_belso);
      exit (0);
    }
  else if (gyermekem_pid > 0)
    {
      wait (&statusz);
    }
  else
    {
      exit (-1);
    }
  printf ("SZ: %d %d %d %d\n", kulso, statikus_kulso, belso, statikus_belso);
}

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

Mit ír ki? (melyik alábbi 4 sort?)

GY: 1 1 1 1
SZ: 0 0 0 0
GY: 1 1 1 1
SZ: 0 0 0 0

GY: 1 1 1 1
SZ: 1 1 0 0
GY: 1 1 1 1
SZ: 1 1 0 0

GY: 1 1 1 1
SZ: 1 1 0 0
GY: 2 2 1 1
SZ: 2 2 0 0

GY: 1 1 1 1
SZ: 0 0 0 0
GY: 2 2 2 2
SZ: 0 0 0 0

Laborkártyák

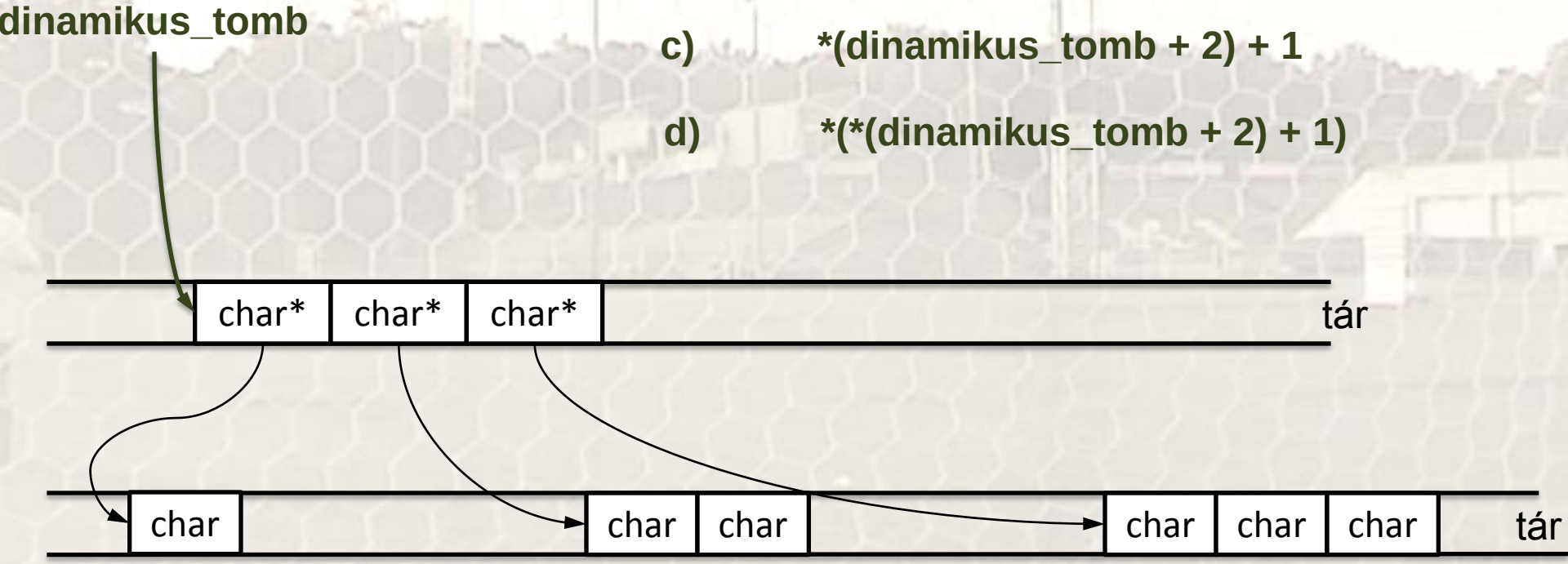
char ** dinamikus_tomb;

a) dinamikus_tomb + 2

b) *(dinamikus_tomb + 2)

c) *(dinamikus_tomb + 2) + 1

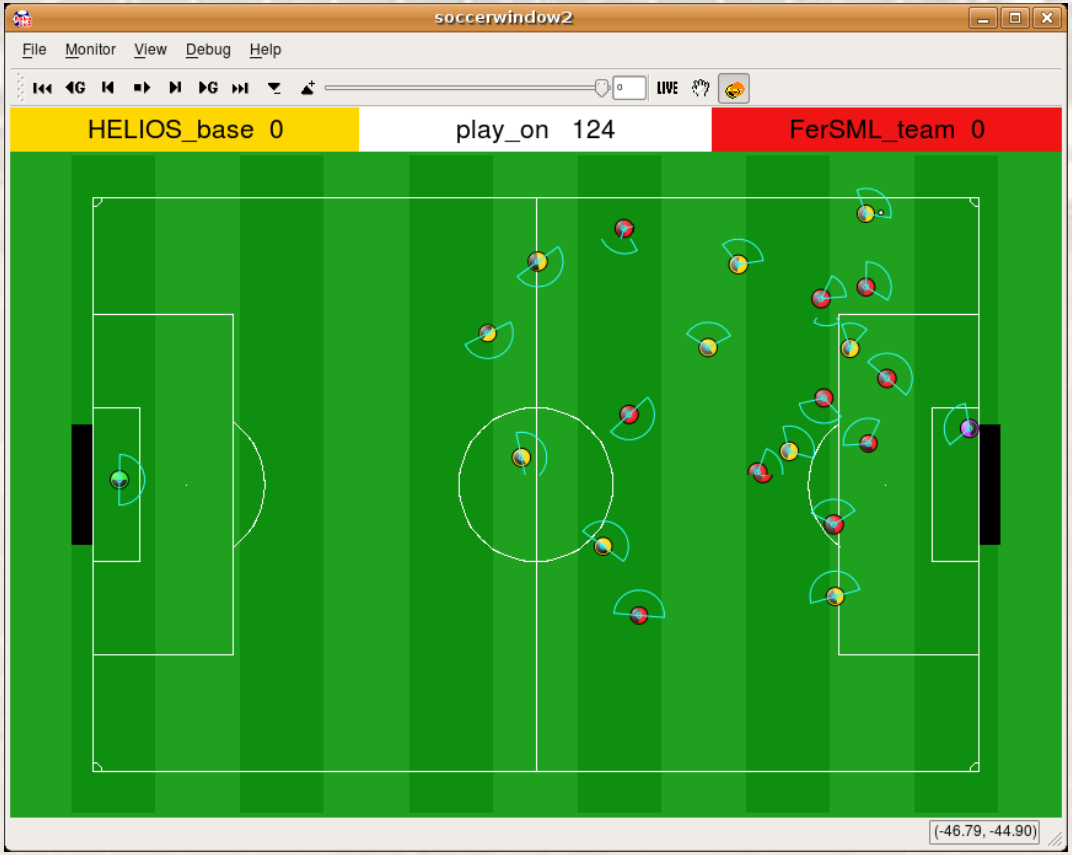
d) *(*(dinamikus_tomb + 2) + 1)



Mi az a, b, c, d? Mutasd az ábrán!

Otthoni opcionális feladat

A PP 41-44 oldali példák kipróbálása, a sigaction kapcsán a robotfocis http://fersml.blog.hu/2011/01/05/ismerkedes_a_japan_helios_csapat_szoftvereivel_avagy_nehany_trivialis_hello_vilag kipróbálása.



Kötelező olvasmány

K&R könyvből olvassuk el (többször!) a negyedik és ötödik fejezetet:

a) Függvények és programstruktúra
b) Mutatók és tömbök

Algoritmikus kérdésekre (megállási probléma, Kolmogorov bonyolultság) a Javát tanítok http://www.tankonyvtar.hu/informatika/javat-tanitok-javat-080904 mellett a Rónyai-Iványos-Szabó Algoritmusok című könyv megfelelő részeiből történő mazsolázás adhatja a tökéletes élményt.