MYY502

MYY502

MYY502

Η γλώσσα C

Δείκτες και Διαχείριση Μνήμης

(memory management)

1 πείραμα = πολλές μετρήσεις

- Κατασκευάστε ένα πρόγραμμα το οποίο δέχεται ως είσοδο από το χρήστη τον συνολικό αριθμό μετρήσεων που έγιναν κατά τη διάρκεια ενός πειράματος
- Εν συνεχεία ο χρήστης εισάγει στο πρόγραμμα τις επιμέρους τιμές των μετρήσεων σε ένα πίνακα
- Τέλος το πρόγραμμα επεξεργάζεται τα δεδομένα
  - μεταξύ άλλων υπολογίζει τον μέσο όρο των μετρήσεων

Άγνωστο μέγεθος πίνακα

```
#include <stdio.h>
int main() {
   int n;
   int i;
   float mo = 0;

   scanf("%d", &n);
   float measurements[n];

   for(i=0; i<n; i++)
       scanf("%f", &measurements[i]);

   for(i=0; i<n; i++)
       mo += measurements[i];

   mo = mo / n;
   ...
}
```

πώς σας φαίνεται αυτή η υλοποίηση?

Άγνωστο μέγεθος πίνακα

```
#include <stdio.h>
int main() {
   int n;
   int i;
   float mo = 0;

   scanf("%d", &n);
   float measurements[n];

   for(i=0; i<n; i++)
       scanf("%f", &measurements[i]);

   for(i=0; i<n; i++)
       mo += measurements[i];

   mo = mo / n;
   ...
}
```

ΌΧΙ ΚΑΙ ΤΟΣΟ ΚΑΛΗ….

α) οι δηλώσεις καλό είναι να γίνονται στην αρχή του κάθε μπλοκ { … }

β) κυρίως, όμως, ΔΕΝ επιτρέπεται να είναι μεταβλητό το μέγεθος του πίνακα!

Δυναμική δέσμευση μνήμης (dynamic memory allocation)

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int n; int i; float mo = 0;
   float *measurements;
   scanf("%d", &n);

   /* Μνήμη για n μετρήσεις */
   measurements = (float *) malloc(n * sizeof(float));
   if (measurements == NULL) exit(1);
   for(i=0; i<n; i++)
       scanf("%f", &measurements[i]);

   for(i=0; i<n; i++)
       mo += measurements[i];

   mo = mo / n;
   free (measurements);
   ...
}
```

malloc

- Η συνάρτηση malloc:

```
int *p;
p = (int *) malloc(N*sizeof(int));
```

- Η `malloc` δεσμεύει μνήμη δοσμένου μεγέθους και επιστρέφει μια διεύθυνση
- Όρισμα = το μέγεθος της αιτούμενης μνήμης (# bytes)
  - **π.χ., στην πράξη θα χρειαστούμε μνήμη για αποθήκευση `int`, μνήμη για αποθήκευση `float`, μνήμη για αποθήκευση `double`,…**
  - **Το μέγεθος της μνήμης δίνεται ως πολλαπλάσιο** των `sizeof(int), sizeof(float), sizeof(double)`,…

Δυναμική δέσμευση μνήμης

- Η `malloc` δεν μπορεί να επιστρέφει διαφορετικούς τύπους από pointers
  - Επιστρέφει έναν «γενικό» τύπο pointer:
    ```
    void *malloc(int size);
    ```
- Επομένως, είναι πάντα απαραίτητο να χρησιμοποιούμε το κατάλληλο casting, π.χ. (`int *`), για αυτό που επιστρέφει η `malloc`.

- Άρα αν για ένα μονοδιάστατο πίνακα τύπου `<T>` (όπου `<T>` είναι `int`, `float`, `double`, `char`…) δεν ξέρω τη διάσταση του πριν την εκτέλεση του προγράμματος,
- τότε ορίζω δείκτη `<T*> p` και χρησιμοποιώ την `malloc`

```
p = (<T*>) malloc(N*sizeof(<T>));
```

Π.χ.

```
float *p;
p = (float *) malloc(N*sizeof(float));
```

Παρένθεση - υπενθύμιση

- Το a και β παρακάτω είναι ίδια; Αν όχι υπάρχει κάποιο πρόβλημα;

(a)	(b)
`int x, *y = &x;`	`int x, *y = NULL;` `*y = &x;`

- Θυμηθείτε ότι άλλο σημαίνουν οι τελεστές της C μέσα σε μία ΔΗΛΩΣΗ και άλλο μέσα σε μία πράξη. Το (b) λοιπόν είναι διαφορετικό (και λάθος). Το (a) θα ήταν ισοδύναμο με το (c) – *να προτιμάτε πάντα το (c):*

(a)	(c)
`int x, *y = &x;`	`int x, *y = NULL;` `y = &x;`

- Επομένως, η σωστή αρχικοποίηση ενός δείκτη με malloc() είναι μία από τις δύο παρακάτω – *να προτιμάτε πάντα το (b):*

(a)	(b)
`int *y = (int *) malloc(40);`	`int *y = NULL;` `y = (int *) malloc(40);`

malloc & free

- Η μνήμη που δεσμεύουμε μέσω της `malloc()`:
  - Δεν είναι αρχικοποιημένη (περιέχει τυχαίες τιμές)
  - Αποδεσμεύεται αυτόματα στο τέλος του προγράμματος και επιστρέφεται στο λειτουργικό σύστημα
  - Επομένως, όσο ζητάμε νέους χώρους μνήμης, τόσο μειώνεται η ελεύθερη μνήμη του συστήματος (θέλει προσοχή ώστε να μην φτάσουμε σε “out of memory”)

- Μπορούμε να αποδεσμεύσουμε / απελευθερώσουμε μνήμη με τη συνάρτηση free:

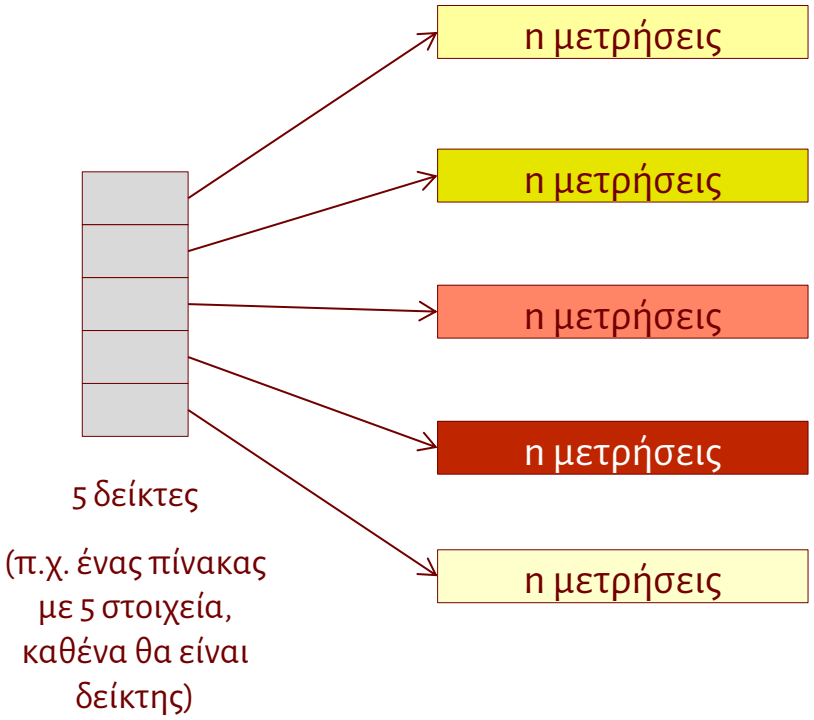
```
void free(void *ptr);
```

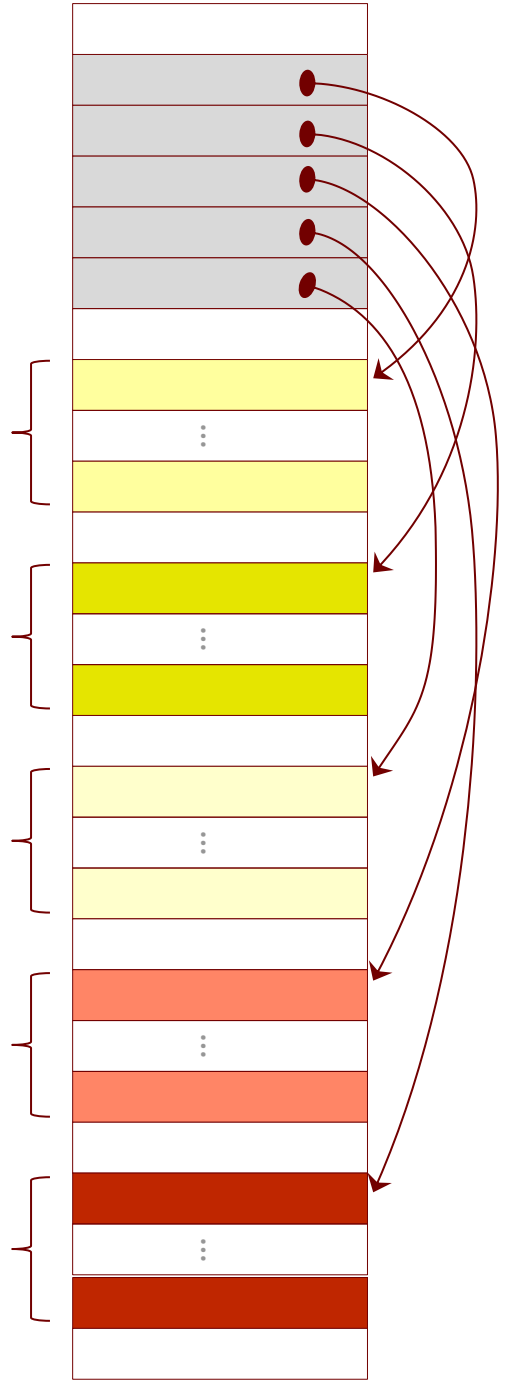
όπου ο `ptr` δείχνει στην αρχή του χώρου που πριν τον είχαμε δεσμέυσει με `malloc()`.

5 πειράματα με n μετρήσεις

- Κατασκευάστε ένα πρόγραμμα το οποίο δέχεται ως είσοδο από το χρήστη τον συνολικό αριθμό μετρήσεων που έγιναν κατά τη διάρκεια 5 πειραμάτων
- εν συνεχεία ο χρήστης εισάγει στο πρόγραμμα τις επιμέρους τιμές των μετρήσεων για κάθε πείραμα
- τέλος το πρόγραμμα επεξεργάζεται τα δεδομένα
  - μεταξύ άλλων υπολογίζει το μέσο όρο των μετρήσεων

Σχέδιο μνήμης





Δυναμική δέσμευση μνήμης & δείκτες

```c
#include <stdio.h>
#include <stdlib.h>
#define M 5

int main() {
  int n, i,j; float mo;
  float *measurements[M];   /* Πίνακας από Μ δείκτες / ίδιο με float *(measurements[M]); */

  scanf("%d", &n);
  for(i=0; i<M; i++){                          /* Για κάθε πείραμα, μνήμη για n μετρήσεις */
    measurements[i] = (float *) malloc(n*sizeof(float));    /* Δείχνει σε χώρο n αριθμών */
    if (measurements[i] == NULL) return 1;
  }

  for(i=0; i<M; i++)
    for(j=0; j<n; j++)
      scanf("%f", &measurements[i][j]);    /* Χειρισμός σαν να είναι διδιάστατος πίνακας */
  for(i=0; i<M; i++) {
    for(j=0,mo=0.0; j<n; j++)
      mo += measurements[i][j];
    mo = mo / n;
    printf("%f\n", mo);
  }

  for(i=0; i<M; i++)
    free (measurements[i]);               /* Απελευθέρωση μνήμης */
  return 0;
}
```

Ερωτήσεις κρίσεως στην προηγούμενη διαφάνεια

- Πώς μπορώ να γράψω την παρακάτω έκφραση κάνοντας χρήση μόνο δεικτών και πράξεων (όχι αγκυλών)

```
if (measurements[i] == NULL) return 1;
```

- Έτσι:

```
if (*(measurements+i) == NULL) return 1;
```

- Το παρακάτω;

```
mo = measurements[i][j];
```

- Έτσι:

```
mo = *( *(measurements+i) + j );
```

- Διότι είναι ισοδύναμο με:

```
mo = *( measurements[i] + j );
```

- το οποίο είναι ισοδύναμο με:

```
mo = ( measurements[i] )[j];
```

Πολλά πειράματα, καθένα πολλές μετρήσεις

(Άγνωστος ο αριθμός των πειραμάτων εκ των προτέρων)

- Κατασκευάστε ένα πρόγραμμα το οποίο δέχεται ως είσοδο από τον χρήστη
  - τον συνολικό αριθμό πειραμάτων που έγιναν
  - τον συνολικό αριθμό μετρήσεων που έγιναν κατά τη διάρκεια ενός πειράματος
- εν συνεχεία ο χρήστης εισάγει στο πρόγραμμα τις επιμέρους τιμές των μετρήσεων για κάθε πείραμα
- τέλος το πρόγραμμα επεξεργάζεται τα δεδομένα
  - μεταξύ άλλων υπολογίζει το μέσο όρο των μετρήσεων

Δυναμική δέσμευση μνήμης & δείκτες

- Στο προηγούμενο πρόβλημα γνωρίζαμε των αριθμό των πειραμάτων (`#define M 5`) και με βάση τον αριθμό των μετρήσεων (`int n`) κατασκευάσαμε έναν πίνακα από δείκτες, καθένας εκ των οποίων έδειχνε σε μια «γραμμή» από n μετρήσεις
  - Το χειριζόμαστε σαν να είναι διδιάστατος πίνακας `measurements[5][n]`
- Στο πρόβλημα αυτό πρέπει να κατασκευάσουμε δυναμικά και τον πίνακα από τους δείκτες, μιας και τόσο το πλήθος των μετρήσεων ανά πείραμα (n) όσο και το πλήθος των πειραμάτων (m) είναι μεταβλητές του προγράμματος.

Παράδειγμα

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

int main() {
  int n,m, i, j;
  float mo;
  float **measurements;

  scanf("%d", &m);     /* Μαθαίνουμε τις διαστάσεις */
  scanf("%d", &n);

  /* Μνήμη για m δείκτες σε πειράματα */
  measurements = (float **) malloc( m*sizeof(float *) );
  if (measurements == NULL)
    return 1;

  for(i=0; i<m; i++) {  /* Για κάθε πείραμα, μνήμη για n μετρήσεις */
    measurements[i] = (float *) malloc(n*sizeof(float));
    if (measurements[i] == NULL)
      return 1;
  }
```

Παράδειγμα

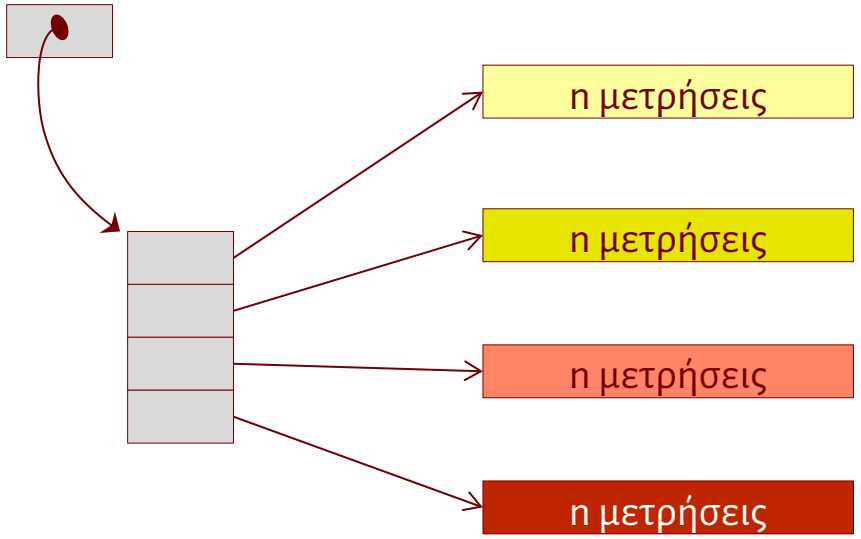
```
  for(i=0; i<m; i++)
    for(j=0; j<n; j++) {
      scanf("%f", &measurements[i][j]);
    }


  for(i=0; i<m; i++) {
    for(j=0, mo=0.0; j<n; j++)
      mo += measurements[i][j];
    mo = mo / n;
    printf("%f\n", mo);
  }

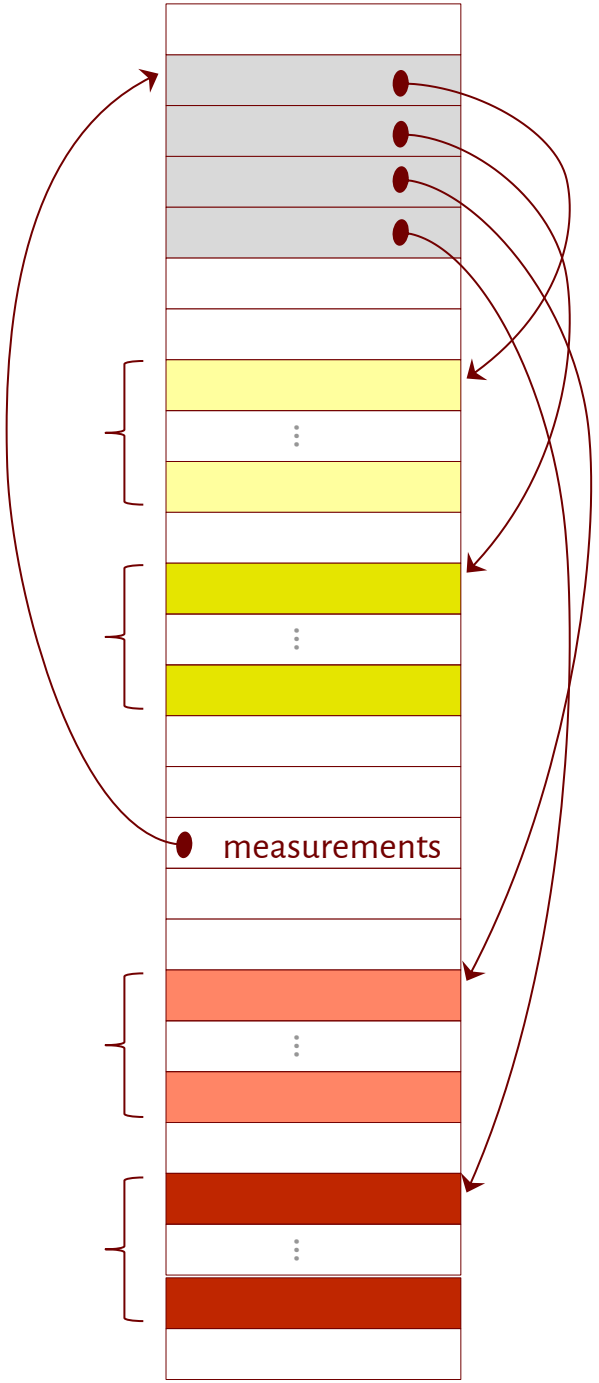
  for(i=0; i<m; i++)
    free(measurements[i]);          /* Απελευθέρωση γραμμής i */
  free(measurements);               /* Απελευθέρωση του πίνακα δεικτών */

  return 0;
}
```

Σχεδιάγραμμα

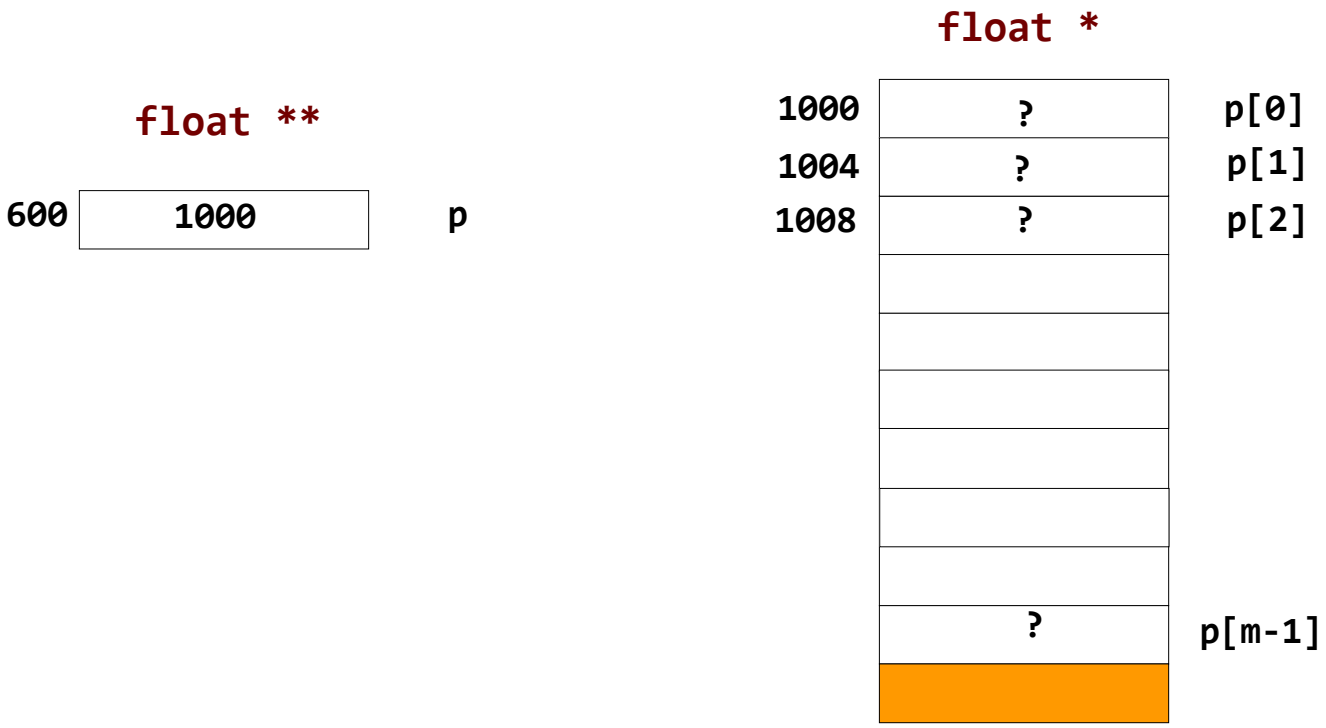


πίνακας με m=4 στοιχεία, καθένα θα είναι δείκτης



Δυναμική δέσμευση και πίνακες δεικτών

```
float **p = NULL;
p = (float **) malloc(m*sizeof(float *));
```

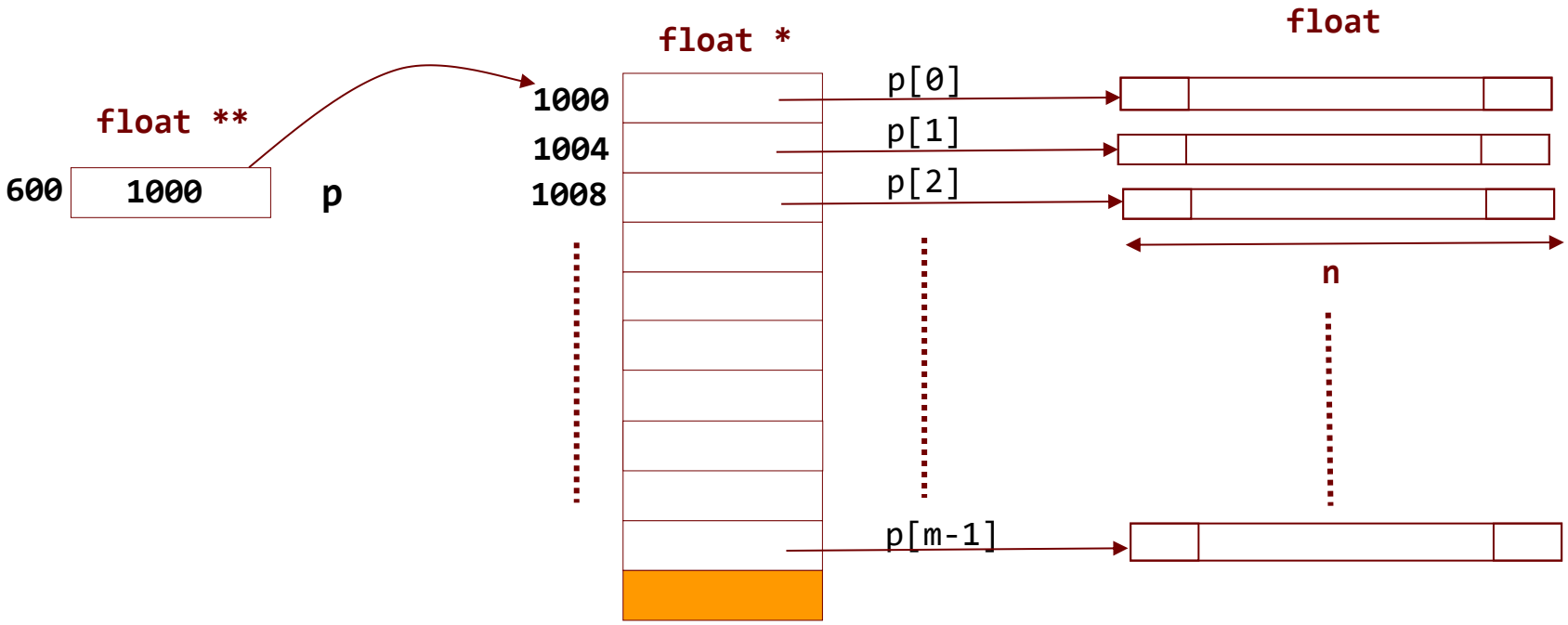


Δυναμική δέσμευση και πίνακες δεικτών

- Αν στη συνέχεια κάνω:

```
p[i] = (float *) malloc(n*sizeof(int));
```

- … θα έχω δημιουργήσει κάτι σαν «διδιάστατο» πίνακα (mxn), όπως τους φτιάχνει και η Java:



Προηγούμενο παράδειγμα – κλήση σε συνάρτηση

```
  for(i=0; i<m; i++)
    for(j=0; j<n; j++) {
      scanf("%f", &measurements[i][j]);
    }

  show_array(m, n, measurements);  /* Πώς ορίζεται; */

  for(i=0; i<m; i++) {
    for(j=0, mo=0.0; j<n; j++)
      mo += measurements[i][j];
    mo = mo / n;
    printf("%f\n", mo);
  }

  for(i=0; i<m; i++)
    free(measurements[i]);          /* Απελευθέρωση γραμμής i */
  free(measurements);               /* Απελευθέρωση του πίνακα δεικτών */

  return 0;
}
```

Παράμετροι σε συνάρτηση

```
void show_array(int r, int c, ?????)
  int i, j;
  for (i = 0; i < r; i++) {
     for (j = 0; j < c; j++)
        printf("%f\n", arr[i][j]);
     printf("\n");
  }
}
```

```
void show_array(int r, int c, float arr[r][c])
```

- Στη C δεν επιτρέπονται πίνακες μεταβλητού μεγέθους

```
void show_array(int r, int c, float arr[][])
```

- Και δεν επιτρέπεται αλλά και δεν γνωρίζει το μέγεθος των γραμμών

```
void show_array(int r, int c, float *arr[])
```

- Μια χαρά! Πίνακας από pointers!

```
void show_array(int r, int c, float **arr)
```

- Μια χαρά! Pointer που δείχνει σε pointers (δηλ. στον πρώτο pointer ενός πίνακα από pointers).

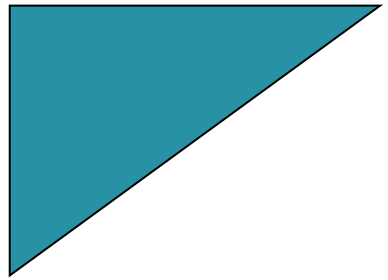
```
void show_array(int r, int c, float (*arr)[])
```

- Λάθος! Pointer σε πίνακα από floats.

Παραδείγματα

Α) Δημιουργία **τριγωνικού** πίνακα με 10 γραμμές.

- Δηλαδή η γραμμή 0 θα έχει 10 στοιχεία, η γραμμή 1 θα έχει 9 στοιχεία, κλπ, και η γραμμή 9 θα έχει 1 στοιχείο.



Β) Δημιουργία τριγωνικού πίνακα με το χρήστη να καθορίζει κατά το χρόνο εκτέλεσης τον αριθμό των γραμμών

Τριγωνικός πίνακας 10x10

```
#include <stdio.h>
#include <stdlib.h>
#define N 10

int main() {
   int i, j, *a[N]; /* Πίνακας με N=10 δείκτες, σε N γραμμές */


   for (i = 0; i < N; i++)
       a[i] = (int *)malloc((N-i)*sizeof(int)); /* Γραμμή i */
   for (i = 0; i < N; i++) {
       for (j = 0; j < N-i; j++) {
              a[i][j] = i + j;
              printf("a(%d,%d)=%d ", i, j, a[i][j]);
       }
       printf("\n");
   }
    return 0;
}
```

Τριγωνικός πίνακας NxN

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int i, j, N, **a; /* Θα δείχνει σε N δείκτες */

   scanf("%d", &N);
   a = (int **) malloc(N*sizeof(int *));
   if (a == NULL) exit(1);
   for (i = 0; i < N; i++)
       a[i] = (int *)malloc((N-i)*sizeof(int)); /* Γραμμή i */
   for (i = 0; i < N; i++) {
       for (j = 0; j < N-i; j++) {
              a[i][j] = i + j;
              printf("a(%d,%d)=%d ", i, j, a[i][j]);
       }
       printf("\n");
   }
    return 0;
}
```

Πώς δεσμεύω καθαρά διδιάστατους πίνακες;

- Δηλαδή όχι απλά πίνακα από pointers που δείχνουν σε M σκόρπιες γραμμές (τον οποίο τον χειρίζομαι μεν σαν να είναι διδιάστατος, όμως δεν καταλαμβάνει συνεχόμενο χώρο στη μνήμη)
- **αλλά** ένα πίνακα που όντως καταλαμβάνει συνεχόμενο χώρο M*N στοιχείων στην μνήμη;

Απάντηση:

- Δεν γίνεται – η malloc() επιστρέφει πάντα ένα συνεχόμενο γραμμικό χώρο.
- Μπορώ όμως να τον χειριστώ εγώ «με το χέρι».
- Αν δεσμεύσω `a = (int *) malloc(M*N*sizeof(int));` ποιο είναι το στοιχείο στην i γραμμή και j στήλη;
  - Το `a[i*N + j]`.

Χειρισμός διδιάστατου πίνακα με malloc()

```
void func() {
  int i, j, M, N;
  int *a;

  scanf("%d", &M);          /* Γραμμές */
  scanf("%d", &N);          /* Στήλες */
  a = (int *) malloc(M*N*sizeof(int));

  for (i = 0; i < M; i++)
     for (j = 0; j < N; j++)
          a[i*N+j] = 1;

  free(a);                  /* 1 free για όλο το χώρο */
}
```

Επιλογή τύπου πίνακα 2D

- Όταν δεν μου ζητείται **ρητά** ο ένας τύπος ή ο άλλος (δηλαδή καθαρά διδιάστατος ή «γιαλαντζί» διδιάστατος – πίνακας από δείκτες σε σκόρπιες γραμμές), τι κάνω;
- Εξαρτάται από την εφαρμογή.
  - Π.χ. αν καλείται κάποια συνάρτηση που περιμένει ως παράμετρο έναν (πραγματικό) διδιάστατο τότε είμαι ***αναγκασμένος*** να δεσμεύσω χώρο για έναν τέτοιον πίνακα και να τον χειριστώ όπως δείξαμε.
  - Π.χ. αν οι γραμμές δεν έχουν ίδιο μέγεθος, αναγκαστικά θα πρέπει να έχω τον δεύτερο τύπο.
  - Αν δεν υπάρχει περιορισμός, είναι συνήθως πιο καλό / εύκολο / ευέλικτο να έχω έναν πίνακα από pointers σε σκόρπιες γραμμές,
  - παρά το γεγονός ότι ένας τέτοιος πίνακας έχει λίγο παραπάνω κόπο στο να δημιουργηθεί (πρέπει να κάνω `malloc` κάθε γραμμή ξεχωριστά) και να ελευθερωθεί (πρέπει να κάνω `free` κάθε γραμμή ξεχωριστά).

Ταξινόμηση strings (το πλήθος δίνεται ως όρισμα στη main) I

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void strBubbleSort(char *strings[], int num);   /* Πρωτότυπο */
#define SIZE 30             /* Μέγιστο μήκος για κάθε string */

int main(int argc, char *argv[]) {
   int  i, NumOfStrings;
   char **array;                  /* Για αποθήκευση των strings */

   if (argc < 2) return 1;        /* ή exit(1) */
   NumOfStrings = atoi(argv[1]);
   if (NumOfStrings < 1) return 1;

   /* Μνήμη για τον πίνακα */
   array = (char **) malloc(NumOfStrings*sizeof(char*));
   if (array == NULL) {
        printf("no memory!\n"); return 1;
   }
   /* Διάβασμα των string */
   for (i = 0; i < NumOfStrings; i++) {  /* mallocs & αντιγραφή strings */
        array[i] = (char *) malloc(SIZE*sizeof(char));
        if (array[i] == NULL) {
            printf("no memory!\n"); return 1;
        }
        fgets(array[i], 29, stdin);
   }
```

(Συνέχεια στην επόμενη διαφάνεια)

Ταξινόμηση strings (το πλήθος δίνεται ως όρισμα στη main) II

(Συνέχεια της main())

```
    strBubbleSort(array, NumOfStrings);      /* Ταξινόμηση */
    puts("\n\nThe sorted list in ascending order is:");
    for (i = 0; i < NumOfStrings; i++)
         puts(array[i]);
    return 0;
}

/* Προσέξτε ότι δεν μετακινούνται τα strings – μόνο οι pointers
 */
void strBubbleSort(char *strings[], int num) {
    char *temp;
    int top, seek;

    for (top = 0; top < num-1; top++) {
         for (seek = top+1; seek < num; seek++) {
              if (strcmp(strings[top], strings[seek]) > 0) {
                   temp = strings[seek];          /* Εναλλαγή pointers */
                   strings[seek]= strings[top];
                   strings[top] = temp;
              }
         }
    }
}
```

Άλλες κλήσεις διαχείρισης μνήμης

```
void *calloc(size_t n, size_t size);
```

- Δεσμεύει χώρο στον οποίο αρχικοποιεί όλα τα bytes σε **μηδέν**
- Παίρνει 2 παραμέτρους: το πλήθος των στοιχείων και το μέγεθος του κάθε στοιχείου (σε bytes):
  - `int *p = (int *) calloc(n, sizeof(int));`

```
void *realloc(void *p, size_t size);
```

- Δεσμεύει νέο χώρο και ότι υπήρχε στον παλιό (όπου δείχνει ο p) αντιγράφεται στον νέο.
- Αν το νέο `size` είναι μικρότερο τότε χάνονται κάποια δεδομένα που υπήρχαν στον παλιό (p)

Παράδειγμα

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

int main() {
   int *p, *q;
   p = (int *) malloc(10*sizeof(int));
   if (p == NULL)
       return 0;
   q = (int *) realloc(p, 20*sizeof(int));
   if (q == NULL)
       return 0;
   printf("%p, %p\n", p, q);      /* μπορεί να διαφέρουν */
   return 0;
}
```

Αποδέσμευση μνήμης

```
void free(void *p);
```

Π.χ.

```
p = (int *) malloc(10*sizeof(int));  /* Δέσμευση */
...
free(p);                             /* Αποδέσμευση */
```

- Αποδεσμεύει τον χώρο στον οποίον δείχνει ο p. Ο χώρος αυτός πρέπει να έχει προέλθει από malloc/calloc/realloc.

- Προσοχή:
  - Ο χώρος αποδεσμεύεται (επιστρέφεται στο σύστημα και δεν μπορούμε να τον ξαναχρησιμοποιήσουμε), ΑΛΛΑ
  - ο pointer p ΣΥΝΕΧΙΖΕΙ ΝΑ ΔΕΙΧΝΕΙ ΕΚΕΙ (δεν άλλαξε η τιμή του)!!

Σύνοψη

- `#include <stdlib.h>`
- `void *malloc(size_t size);`
  Δέσμευση μνήμης
- `void *calloc(size_t count, size_t size);`
  Δέσμευση μνήμης αρχικοποιημένης σε μηδενικά bytes
- `void *realloc(void *ptr, size_t size);`
  Επαναδέσμευση μνήμης (επέκταση/συρρίκνωση)
- `void free(void *ptr);`
  Αποδέσμευση μνήμης

Σύνοψη

- ```
  int *p;
  p = (int *) malloc(sizeof(int));
  ```
- ```
  int *p;
  p = (int *) malloc(M*sizeof(int));
  ```
- ```
  int *p[10];
  for (i = 0; i < 10; i++)
     p[i] = (int *) malloc(M*sizeof(int));
  ```
- ```
  int **p;
  p = (int **) malloc(10*sizeof(int *));
  for (i = 0; i < 10; i++)
     p[i] = (int *) malloc(M*sizeof(int));
  ```

Pointer (malloc) από συνάρτηση Ι

```
#include <stdio.h>

void makeArray(int * ip, int size) {
   int i;
   ip  = (int *) malloc(size*sizeof(int));
   if (ip == NULL) return;
   for (i=0; i<size; i++) {
       ip[i] = 3*i+45;
       printf("print from mA: %d\n",ip[i]);
   }
}

int main() {
   int * array = NULL;
   int i, N=12;

   makeArray(array, N);
   for (i=0;i<N;i++)
       printf("print from main: %d\n", array[i]);
    return 0;
}
```

ΔΟΥΛΕΥΕΙ ????

Pointer (malloc) από συνάρτηση II

```
#include <stdio.h>

void makeArray(int **ip, int size) {
   int i;
   *ip = (int *) malloc(size*sizeof(int));
   if (*ip == NULL) return;
   for (i=0; i<size; i++) {
       (*ip)[i] = 3*i+45; //try without ()
       printf("print from mA: %d\n",*((*ip)+i)); //aka (*ip)[i]
   }
}

int main() {
   int * array = NULL;
   int i, N=12;

   makeArray(&array, N);
   for (i=0;i<N;i++)
       printf("print from main: %d\n", array[i]);
    return 0;
}
```

ΔΟΥΛΕΥΕΙ !!!!!

Προβλήματα pointers / διαχ. μνήμης – Dangling pointers

```
int *p;
printf("%d", *p);
```

Το p δεν δείχνει σε καμία θέση (σκουπίδια) - **dangling**

```
int *p;
if (condition) {
  int temp = 1;
  p = &temp;
  printf("%d", *p);
}
printf("%d", *p);
```

Μετά το μπλοκ του if {}, το temp ΔΕΝ ΥΦΙΣΤΑΤΑΙ. Επομένως το p θα δείχνει σε άκυρη / μη νόμιμη θέση - **dangling**

```
int *p;
p = malloc(10*sizeof(int));
if (p == NULL) ...
p[5] = 100;
...
free(p);
...
p[5]--;
```

Μετά το free(), η δεσμευμένη μνήμη από το malloc() ΔΕΝ ΥΠΑΡΧΕΙ. Όμως, το p ΔΕΝ ΕΧΕΙ ΑΛΛΑΞΕΙ! Επομένως δείχνει σε άκυρη θέση - **dangling**

Κίνδυνοι από dangling pointers & προφυλάξεις

- Οι dangling pointers μπορεί να οδηγήσουν σε προσπέλαση άκυρων / παράνομων διευθύνσεων.
  - Ένα πρόβλημα που μπορούν να προκαλέσουν είναι το κρασάρισμα ή χωρίς αιτιολογία δυσλειτουργία του προγράμματος (εξαιρετικά δύσκολο debugging)
  - Ακόμα πιο σοβαρό είναι η πρόκληση ζημιάς / κρασάρισμα ΟΛΟΥ του συστήματος.

- Προφυλάξεις:
  1. *Πάντα αρχικοποιούμε τους pointers κατά τη δήλωσή τους (τουλάχιστον τους θέτουμε ίσους με NULL).*
  2. *Μετά από το free(p), θέτουμε τον p = NULL.*

Έτσι τουλάχιστον αποφεύγουμε ζημιά στο σύστημα αλλά και διευκολύνουμε το debugging μιας και η προσπέλαση σε NULL «χτυπάει» αμέσως.

Προβλήματα pointers / διαχ. μνήμης – Memory leaks

(Δεν έχουν μπει οι έλεγχοι για NULL για απλότητα του κώδικα)

```
char *p;
p = malloc(10*sizeof(char));
scanf("%s", p);
for ( ; *p != '\0'; p++ )
  printf("%c", *p);
...
```

```
int *p, *q;
p = malloc(10*sizeof(int));
q = malloc(20*sizeof(int));
...
if (condition) {
  p = q;
}
printf("%d", p[5]);
…
```

Κανένα πρόβλημα ορθότητας.

Απλά πλέον ο χώρος των 10 θέσεων που δεσμεύσαμε μέσω του p ΔΕΝ ΜΠΟΡΕΙ ΝΑ ΠΡΟΣΠΕΛΑΣΤΕΙ ΜΕ ΤΙΠΟΤΕ.

Ενώ συνεχίζει να υπάρχει στη μνήμη (δεν έχει αποδεσμευθεί), έχει χαθεί ο μοναδικός pointer που γνώριζε τη διεύθυνσή του – **memory leak (διαρροή μνήμης)**

ΜΗΝ ΞΕΧΝΑΤΕ ΤΑ free() !!!