# Διάλεξη 10 - Δείκτες και Πίνακες - 2D+

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

Θανάσης Αυγερινός

# Ανακοινώσεις / Διευκρινήσεις

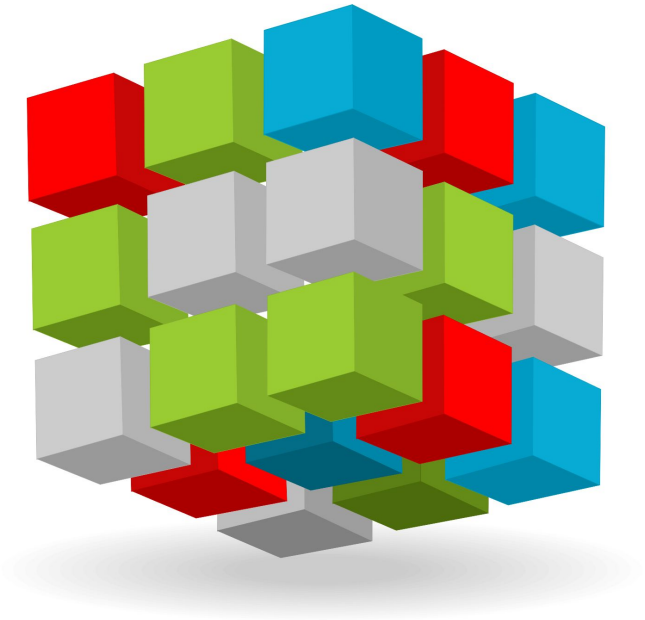
- Να μετατρέψουμε τις ερωτήσεις του Kahoot ή τα προγράμματα που γράφουμε στο μάθημα σε υλικό εξάσκησης
    a. Αν υπάρχει ενδιαφέρον πείτε μου
- Η εργασία #0 είναι πίσω μας, πάμε για την Εργασία #1 τώρα!
- Ενδεικτικές λύσεις για τα εργαστήρια - ευχαριστούμε τον dimskomex!

# Την προηγούμενη φορά

- Δείκτες (Pointers)
  - Διευθύνσεις μνήμης
  - Χρήση δεικτών
  - Πράξεις με δείκτες
  - Παραδείγματα

# Σήμερα

- Δείκτες και Πίνακες reloaded
  - Δισδιάστατοι πίνακες
  - Παραδείγματα
  - Δείκτες σε δείκτες σε δείκτες …
  - Δυναμική διαχείριση μνήμης



# Δήλωση Πίνακα (Array) στην Γλώσσα C

Ένας πίνακας μας επιτρέπει να χειριζόμαστε ένα σύνολο από δεδομένα ίδιου τύπου με ενιαίο και γενικό τρόπο. Στην C η δήλωση ενός πίνακα έχει την μορφή:

```
int array[100];
```

Ο **τύπος (type)** της μεταβλητής λέει στον μεταγλωττιστή πόση μνήμη να δεσμεύσει για κάθε στοιχείο του πίνακα

Το **όνομα (name)** του πίνακα κάνει τον μεταγλωττιστή να διαλέξει την διεύθυνση της μνήμης θα τον αποθηκεύσει

Το **μέγεθος (size)** του πίνακα λέει στον μεταγλωττιστή πόσες θέσεις αυτού του τύπου να κρατήσει - στατικό: αφού δηλωθεί δεν αλλάζει κατά την εκτέλεση

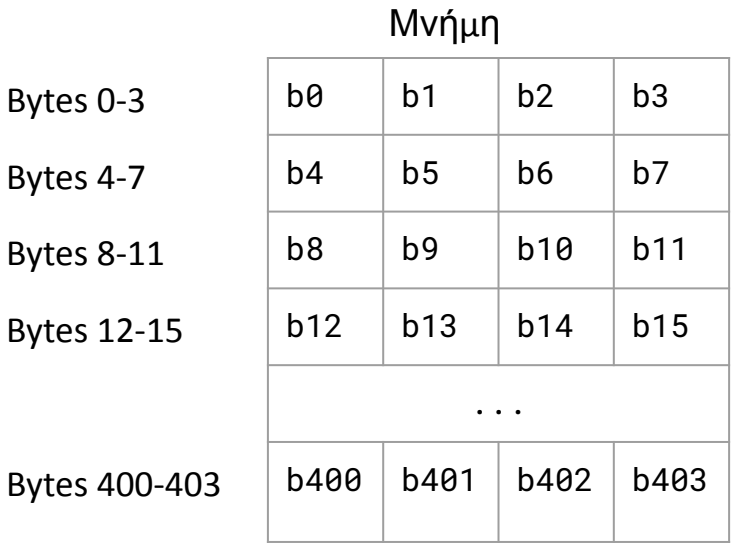
# Δήλωση Πίνακα (Array) στην Γλώσσα C

Ένας πίνακας μας επιτρέπει να χειριζόμαστε ένα σύνολο από δεδομένα ίδιου τύπου με ενιαίο και γενικό τρόπο. Στην C η δήλωση ενός πίνακα έχει την μορφή:

```
int bears[100];
```

Μπορούμε να αναφερθούμε στο κάθε στοιχείο του πίνακα χρησιμοποιώντας την θέση (index) του στοιχείου στον πίνακα:

`bears[0], bears[1], …, bears[99]`



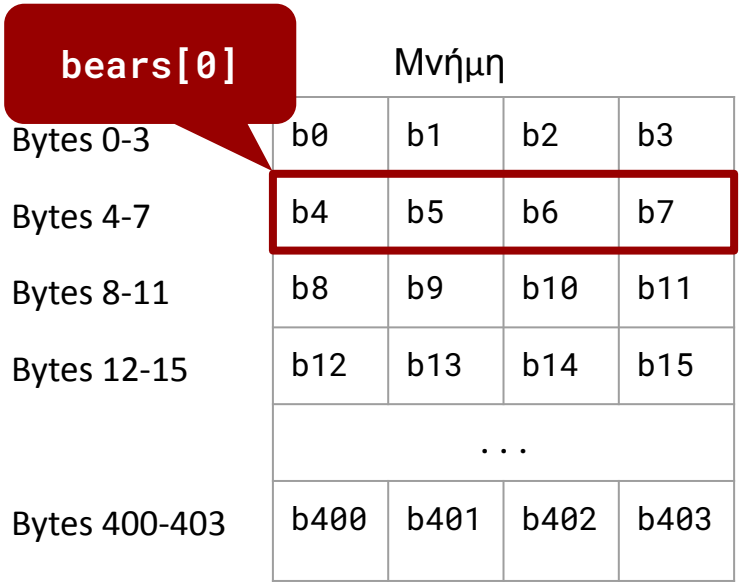
# Δήλωση Πίνακα (Array) στην Γλώσσα C

Ένας πίνακας μας επιτρέπει να χειριζόμαστε ένα σύνολο από δεδομένα ίδιου τύπου με ενιαίο και γενικό τρόπο. Στην C η δήλωση ενός πίνακα έχει την μορφή:

```
int bears[100];
```

Μπορούμε να αναφερθούμε στο κάθε στοιχείο του πίνακα χρησιμοποιώντας την θέση (index) του στοιχείου στον πίνακα:

`bears[0], bears[1], …, bears[99]`



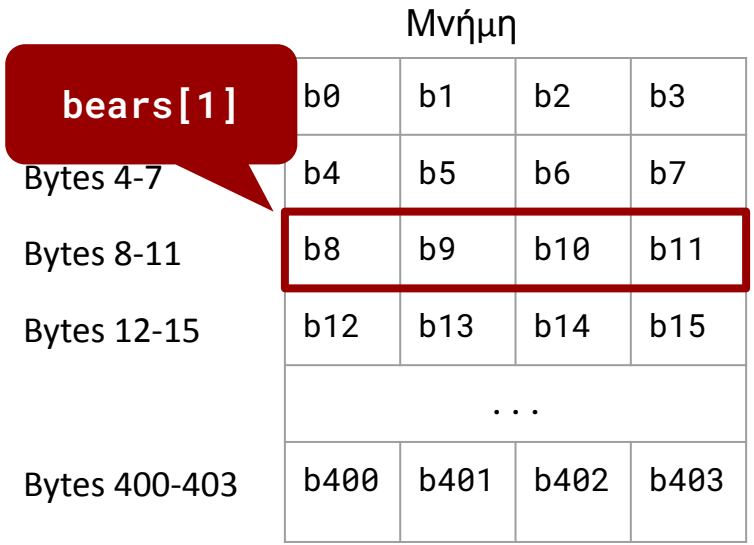
# Δήλωση Πίνακα (Array) στην Γλώσσα C

Ένας πίνακας μας επιτρέπει να χειριζόμαστε ένα σύνολο από δεδομένα ίδιου τύπου με ενιαίο και γενικό τρόπο. Στην C η δήλωση ενός πίνακα έχει την μορφή:

```
int bears[100];
```

Μπορούμε να αναφερθούμε στο κάθε στοιχείο του πίνακα χρησιμοποιώντας την θέση (index) του στοιχείου στον πίνακα:

`bears[0], bears[1], …, bears[99]`



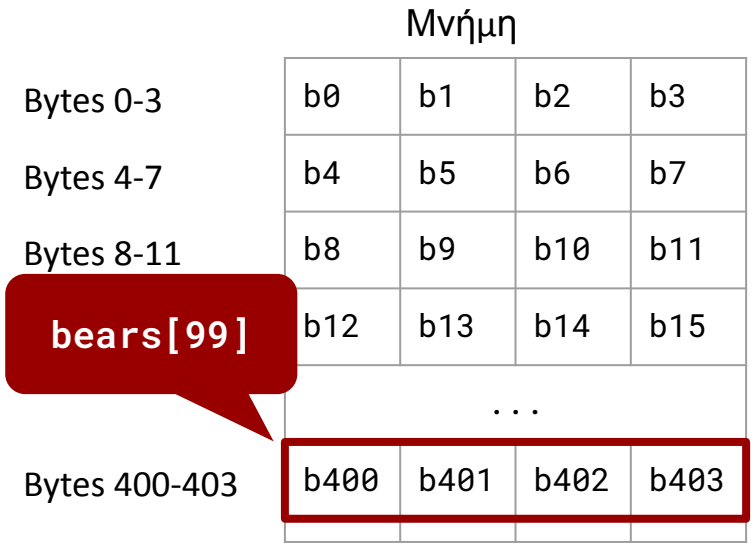
# Δήλωση Πίνακα (Array) στην Γλώσσα C

Ένας πίνακας μας επιτρέπει να χειριζόμαστε ένα σύνολο από δεδομένα ίδιου τύπου με ενιαίο και γενικό τρόπο. Στην C η δήλωση ενός πίνακα έχει την μορφή:

```
int bears[100];
```

Μπορούμε να αναφερθούμε στο κάθε στοιχείο του πίνακα χρησιμοποιώντας την θέση (index) του στοιχείου στον πίνακα:

`bears[0], bears[1], …, bears[99]`



# Δήλωση Πίνακα (Array) στην Γλώσσα C

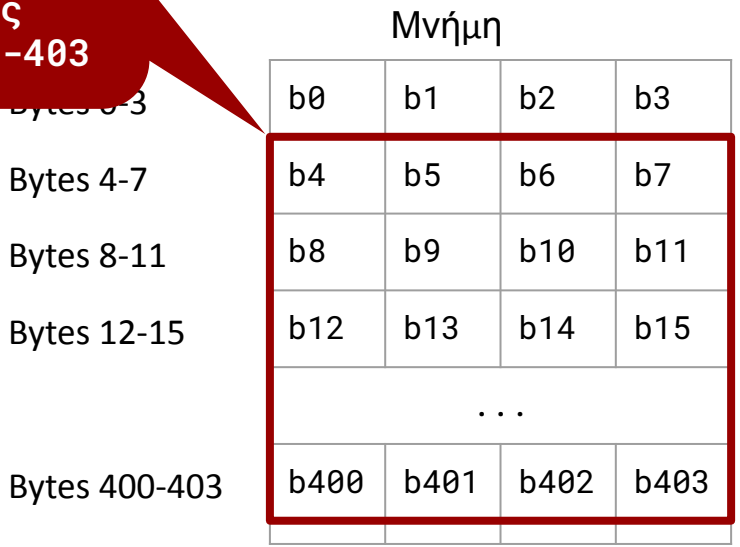
Ένας πίνακας μας επιτρέπει να χειριζόμαστε ένα σύνολο από δεδομένα ίδιου τύπου με ενιαίο και γενικό τρόπο. Στην C η δή[illegible]ν μορφή:

```
int bears[100];
```

Μπορούμε να αναφερθούμε στο κάθε στοιχείο του πίνακα χρησιμοποιώντας την θέση (index) του στοιχείου στον πίνακα:

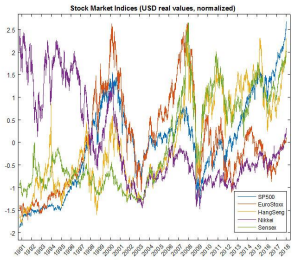
`bears[0], bears[1], …, bears[99]`

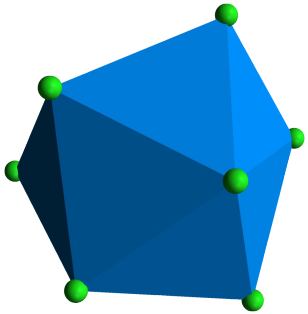
**Ο πίνακας bears καταλαμβάνει 4 * 100 = 400 bytes μνήμης στις διευθύνσεις 4-403**

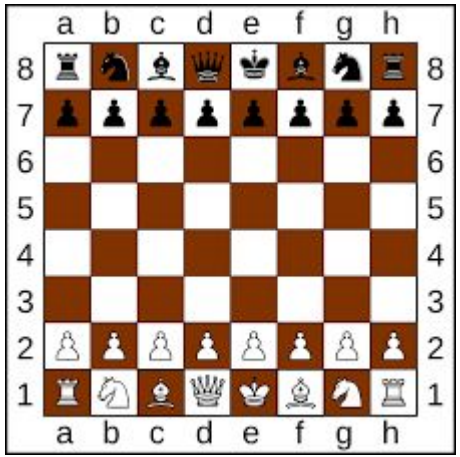


# Κάποια Δεδομένα Αναπαρίστανται Ευκολότερα σε πάνω από μία Διάσταση (Dimension)

- Επεξεργασία εικόνας και γραφικά
- Πολυδιάστατα δεδομένα σε επιστημονικά πεδία (φυσική, χημεία, κτλ)
- Ανάλυση οικονομικών δεδομένων
- Και πολλά άλλα





# Δήλωση Δισδιάστατου Πίνακα (Array) στην Γλώσσα C

Ένας δισδιάστατος πίνακας είναι ένας πίνακας από (υπο)πίνακες. Στην C η δήλωση ενός δισδιάστατου πίνακα έχει την μορφή:

```
τύπος  όνομα[γραμμές][στήλες];
```

Ο **τύπος (type)** περιγράφει τον τύπο κάθε στοιχείου του πίνακα

Το **όνομα (name)** του πίνακα που μας επιτρέπει να αναφερόμαστε σε αυτόν

Ο αριθμός των **γραμμών (rows)** του πίνακα περιγράφει πόσους υποπίνακες (γραμμές) έχει ο πίνακας (1η διάσταση)

Ο αριθμός των **στηλών (columns)** του πίνακα περιγράφει πόσα στοιχεία έχει ο κάθε υποπίνακας (2η διάσταση)

Σύνολο ο πίνακας έχει **γραμμές x στήλες** στοιχεία

# Δήλωση Δισδιάστατου Πίνακα (Array) στην Γλώσσα C

Ένας δισδιάστατος πίνακας είναι ένας πίνακας από (υπο)πίνακες. Ένα παράδειγμα πίνακα 2 γραμμών και 4 στηλών (8 στοιχεία σύνολο) είναι:

```
int array[2][4];
```

Μπορούμε να αναφερθούμε στο κάθε στοιχείο με την έκφραση a[i][j], όπου i είναι η θέση του στοιχείου στην γραμμή και j η θέση του στην στήλη.

| | Στήλη 0 | Στήλη 1 | Στήλη 2 | Στήλη 3 |
|---|---|---|---|---|
| Γραμμή 0 | `a[0][0]` | `a[0][1]` | `a[0][2]` | `a[0][3]` |
| Γραμμή 1 | `a[1][0]` | `a[1][1]` | `a[1][2]` | `a[1][3]` |

# Αρχικοποίηση Δισδιάστατου Πίνακα

Η αρχικοποίηση δισδιάστατων πινάκων είναι όπως στους μονοδιάστατους - αντί για στοιχεία  τύπου, χρησιμοποιούμε πίνακες:

```
int array[2][4] = {
    {1, 4, 7, 10},
    {3, 6, 9, 12},
};
```

Ποιο είναι το στοιχείο `a[1][1];`

| | Στήλη 0 | Στήλη 1 | Στήλη 2 | Στήλη 3 |
|---|---|---|---|---|
| Γραμμή 0 | **1** | **4** | **7** | **10** |
| Γραμμή 1 | **3** | **6** | **9** | **12** |

# Αναπαράσταση Δισδιάστατου Πίνακα στην Μνήμη

Έστω ότι sizeof(int) == 4, τότε μια αναπαράσταση του πίνακα θα είναι:

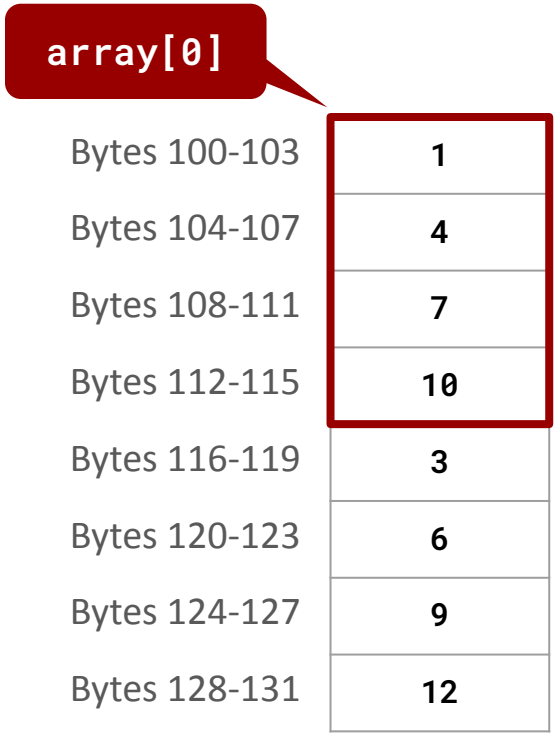
```
int array[2][4] = {
    {1, 4, 7, 10},
    {3, 6, 9, 12},
};
```

| | |
|---|---|
| Bytes 100-103 | 1 |
| Bytes 104-107 | 4 |
| Bytes 108-111 | 7 |
| Bytes 112-115 | 10 |
| Bytes 116-119 | 3 |
| Bytes 120-123 | 6 |
| Bytes 124-127 | 9 |
| Bytes 128-131 | 12 |

# Αναπαράσταση Δισδιάστατου Πίνακα στην Μνήμη

Έστω ότι sizeof(int) == 4, τότε μια αναπαράσταση του πίνακα θα είναι:

```
int array[2][4] = {
    {1, 4, 7, 10},
    {3, 6, 9, 12},
};
```




```
printf("%d\n", sizeof(array[0]));
```

```
$ ./test
16
```

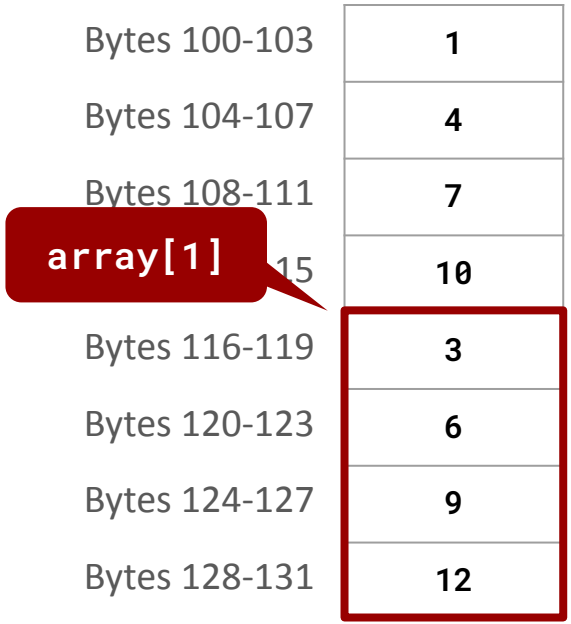
# Αναπαράσταση Δισδιάστατου Πίνακα στην Μνήμη

Έστω ότι sizeof(int) == 4, τότε μια αναπαράσταση του πίνακα θα είναι:

```
int array[2][4] = {
    {1, 4, 7, 10},
    {3, 6, 9, 12},
};
```

```
printf("%d\n", sizeof(array[1]));
```

```
$ ./test
16
```



# Αναπαράσταση Δισδιάστατου Πίνακα στην Μνήμη

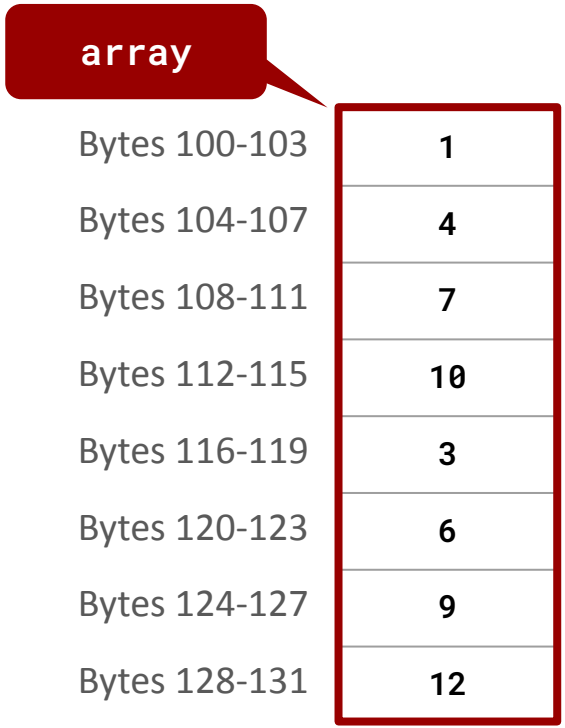
Έστω ότι sizeof(int) == 4, τότε μια αναπαράσταση του πίνακα θα είναι:

```
int array[2][4] = {
    {1, 4, 7, 10},
    {3, 6, 9, 12},
};
```

```
printf("%d\n", sizeof(array));
```

```
$ ./test
```

```
32
```



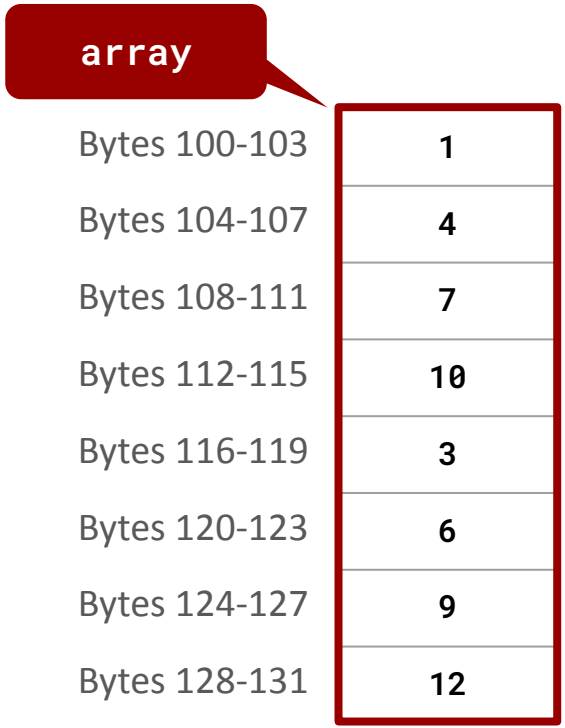
# Υπολογισμός διεύθυνσης στοιχείου στην μνήμη

Έστω η ακόλουθη δήλωση πίνακα:

```
int array[X][Y];
```

Η διεύθυνση του a[i][j] (&a[i][j]) θα είναι:

```
StartAddressOfArray +
```



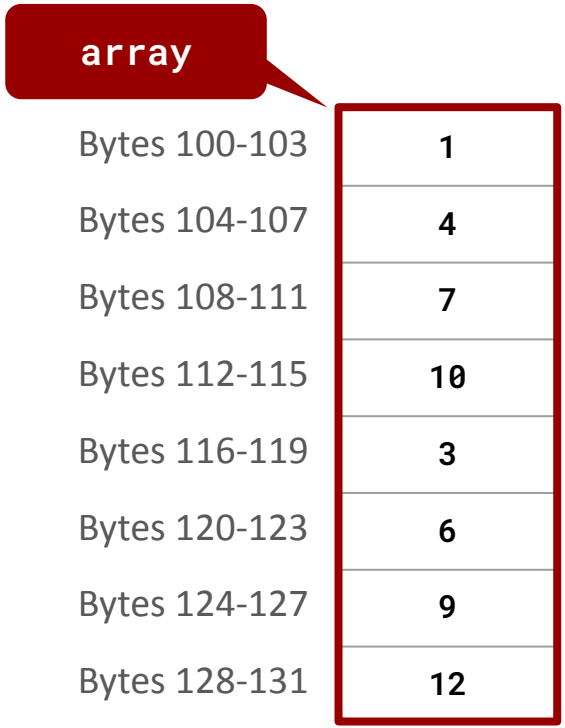
# Υπολογισμός διεύθυνσης στοιχείου στην μνήμη

Έστω η ακόλουθη δήλωση πίνακα:

```
int array[X][Y];
```

Η διεύθυνση του a[i][j] (&a[i][j]) θα είναι:

```
StartAddressOfArray +
    i * Y * sizeof(int) +
    j * sizeof(int)
```



# Υπολογισμός διεύθυνσης στοιχείου στην μνήμη

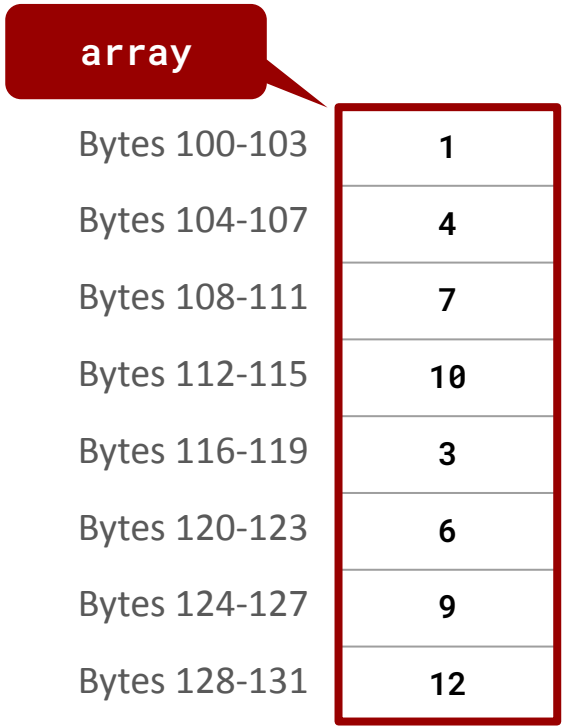
Έστω η ακόλουθη δήλωση πίνακα:

```
int array[X][Y];
```

Η διεύθυνση του a[i][j] (&a[i][j]) θα είναι:

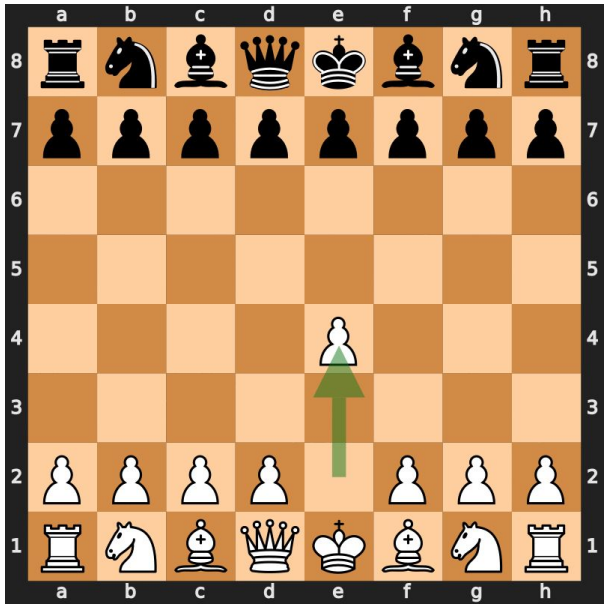
```
IndexOfElementIJ = i * Y + j
```

Παρατηρήστε ότι απλά χρειαζόμαστε τον αριθμό των Y των στηλών για να υπολογίσουμε την θέση του a[i][j]



# Ανάθεση σε Στοιχεία Δισδιάστατου Πίνακα

```
char chessBoard[8][8] = {
    {'R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'},
    {'P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'},
    {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
    {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
    {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
    {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
    {'p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'},
    {'r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'}
};
chessBoard[1][4] = ' ';
chessBoard[3][4] = 'p';
```

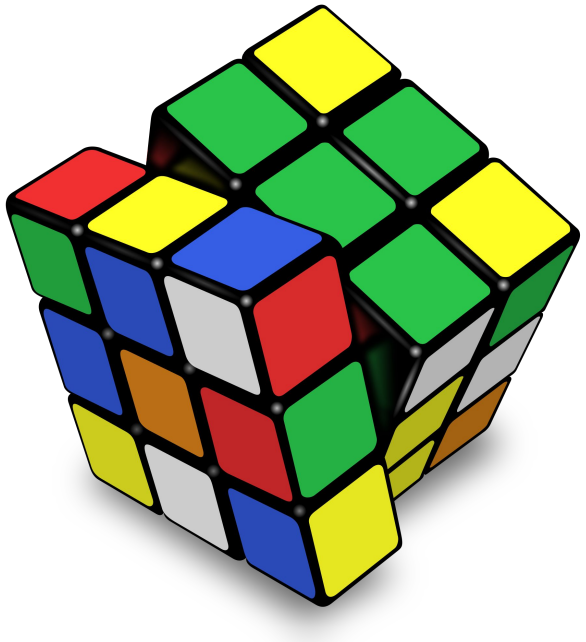



Κάθε array[i][j] στοιχείο μπορεί να χρησιμοποιηθεί ως μεταβλητή

# Τρισδιάστατοι, Τετραδιάστατοι, κοκ

Πίνακες υψηλότερων διαστάσεων λειτουργούν με τον ίδιο τρόπο, προσθέτοντας τον επιθυμητό αριθμό διαστάσεων.

```
int rubiksCube[6][3][3] = {
    {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}}, // Face 1 (e.g., Red)
    {{1, 1, 1}, {1, 1, 1}, {1, 1, 1}}, // Face 2 (e.g., Green)
    {{2, 2, 2}, {2, 2, 2}, {2, 2, 2}}, // Face 3 (e.g., Blue)
    {{3, 3, 3}, {3, 3, 3}, {3, 3, 3}}, // Face 4 (e.g., Yellow)
    {{4, 4, 4}, {4, 4, 4}, {4, 4, 4}}, // Face 5 (e.g., Orange)
    {{5, 5, 5}, {5, 5, 5}, {5, 5, 5}}  // Face 6 (e.g., White)
};
```



Είναι απαραίτητοι οι πολυδιάστατοι πίνακες;

# Χρήση Δεικτών (Dereference Pointers) - Υπενθύμιση

Για να χρησιμοποιήσουμε το περιεχόμενο της μεταβλητής στην οποία δείχνει ένας δείκτης χρησιμοποιούμε τον μοναδιαίο τελεστή * :

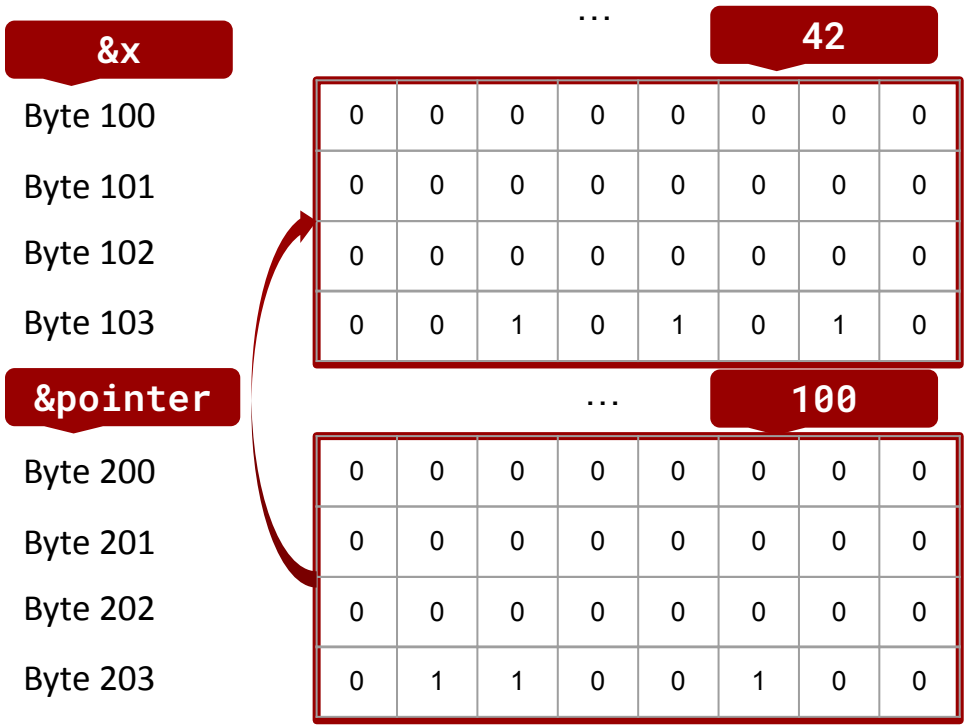
```
int x = 42;
int *pointer = &x;
printf("%d\n", *pointer);
```

Όταν το τρέξουμε:

```
$ ./test

42
```

Η χρήση του `*pointer` είναι ισοδύναμη με την χρήση της μεταβλητής x.



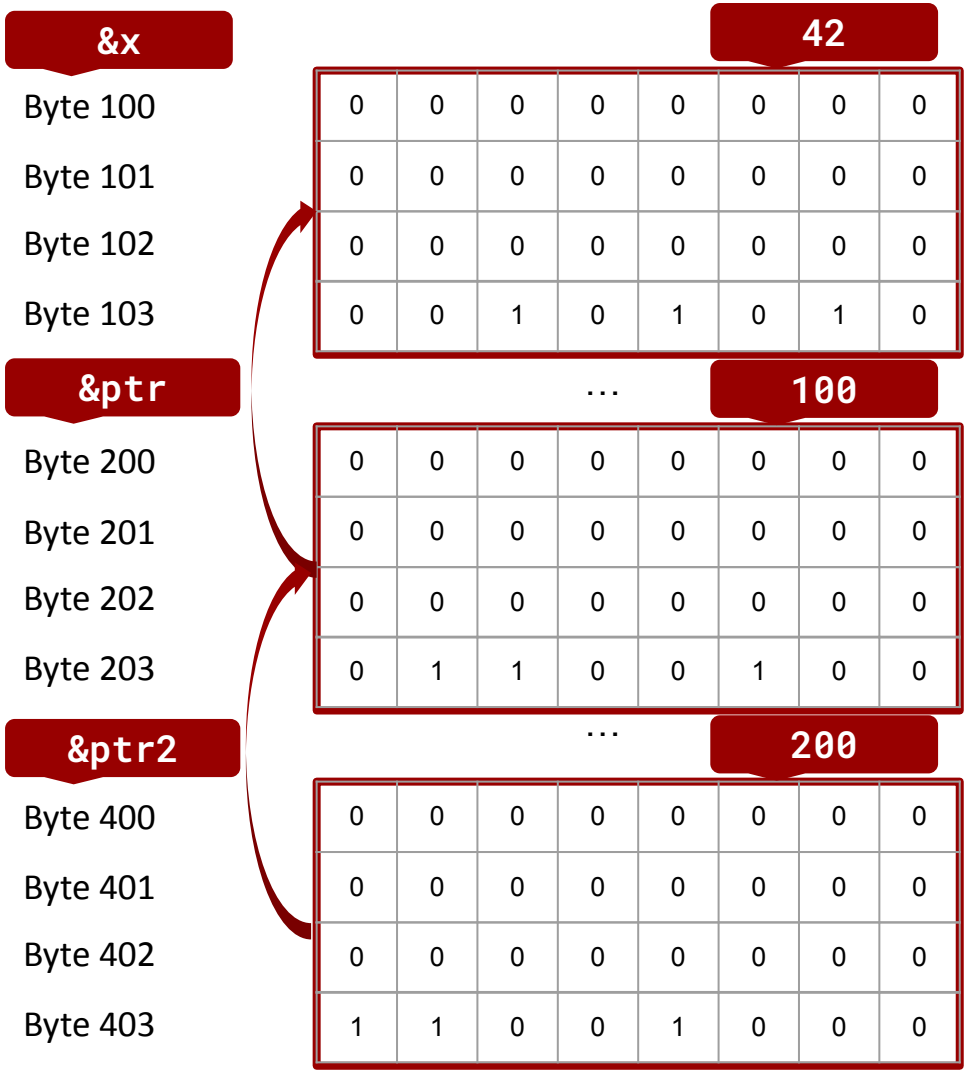
Ποια είναι τα περιεχόμενα του x μετά την εκτέλεση;

```
int * p;
int x[] = {5, 7, 2, 3, 6, 0, 1, 4};
p = x;
while (*p = *(p+2))
  p++;
```

# Δείκτης σε Δείκτη

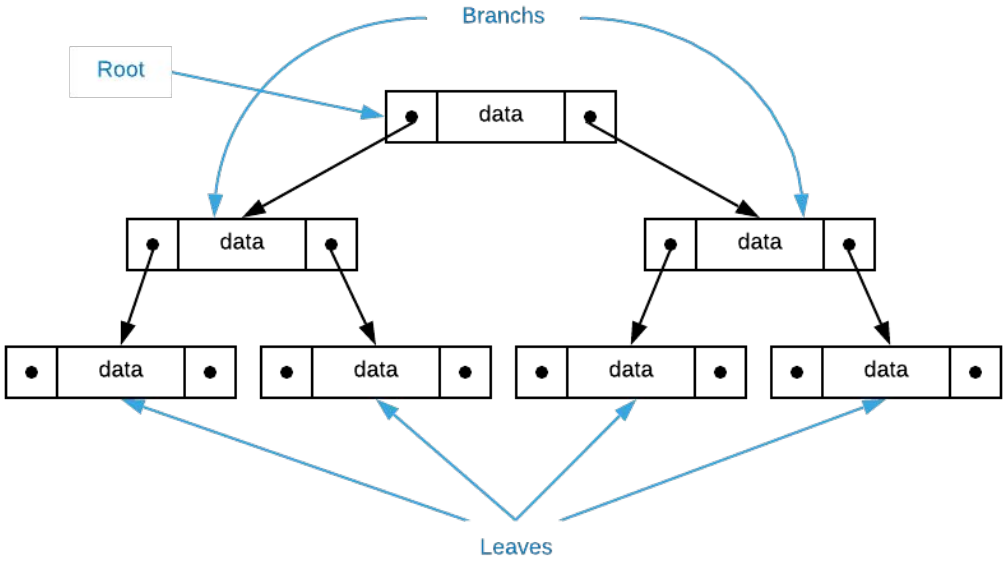
```
int x = 42;
int * ptr = &x;
int ** ptr2 = &ptr;
printf("%p %d", *ptr2, **ptr2);
```

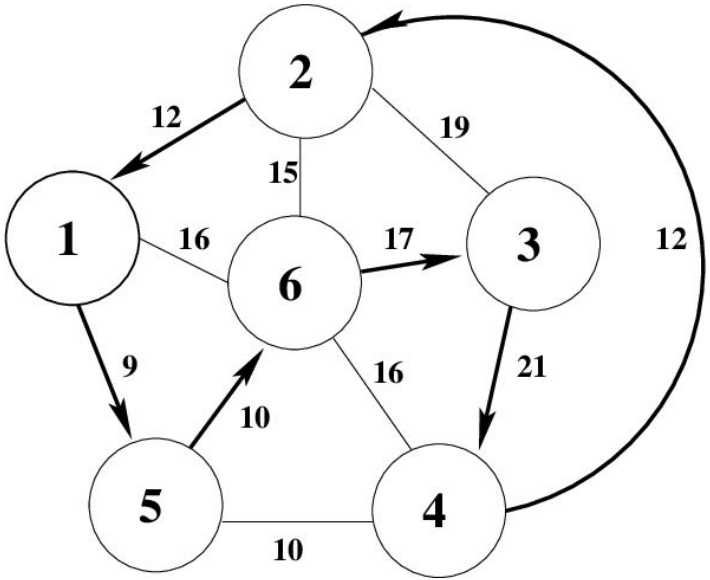
Τι θα τυπώσει το παραπάνω;



# Δείκτες σε Δείκτες σε Δείκτες …

Γιατί έχει σημασία;



# Πίνακες από Δείκτες

Οι δείκτες είναι μεταβλητές και μπορούν να μπουν σε πίνακες. Για παράδειγμα τι θα τυπώσει το ακόλουθο:

```c
#include <stdio.h>
int main() {
  int *ptr[3], a = 100, b = 200, c = 300;
  ptr[0] = &a;
  ptr[1] = &b;
  ptr[2] = &c;
  printf("%d %d %d\n", *ptr[2], *ptr[1], *ptr[0]);
  return 0;
}
```

# Πίνακες από Δείκτες

Οι δείκτες είναι μεταβλητές και μπορούν να μπουν σε πίνακες. Για παράδειγμα τι θα τυπώσει το ακόλουθο:

```c
#include <stdio.h>
int main() {
  int *ptr[3], a = 100, b = 200, c = 300;
  ptr[0] = &a;
  ptr[1] = &b;
  ptr[2] = &c;
  printf("%d %d %d\n", *ptr[2], *ptr[1], *ptr[0]);
  return 0;
}
```

```
$ ./example
300 200 100
```

# Πίνακες από Δείκτες

```c
#include <stdio.h>

int main() {
  char *sentence[] = {
      "I'm", "singing", "in", "the", "rain", "!", NULL
  };
  int i;
  for(i = 0 ; sentence[i]; i++) {
      printf("%s\n", sentence[i]);
  }
  return 0;
}
```

# Πίνακες από Δείκτες

```c
#include <stdio.h>

int main() {

  char *sentence[] = {

      "I'm", "singing", "in", "the", "rain", "!", NULL

  };

  int i;

  for(i = 0 ; sentence[i]; i++) {

      printf("%s\n", sentence[i]);

  }

  return 0;

}
```

```
$ ./sentence
I'm
singing
in
the
rain
!
```

# Το περίφημο argv

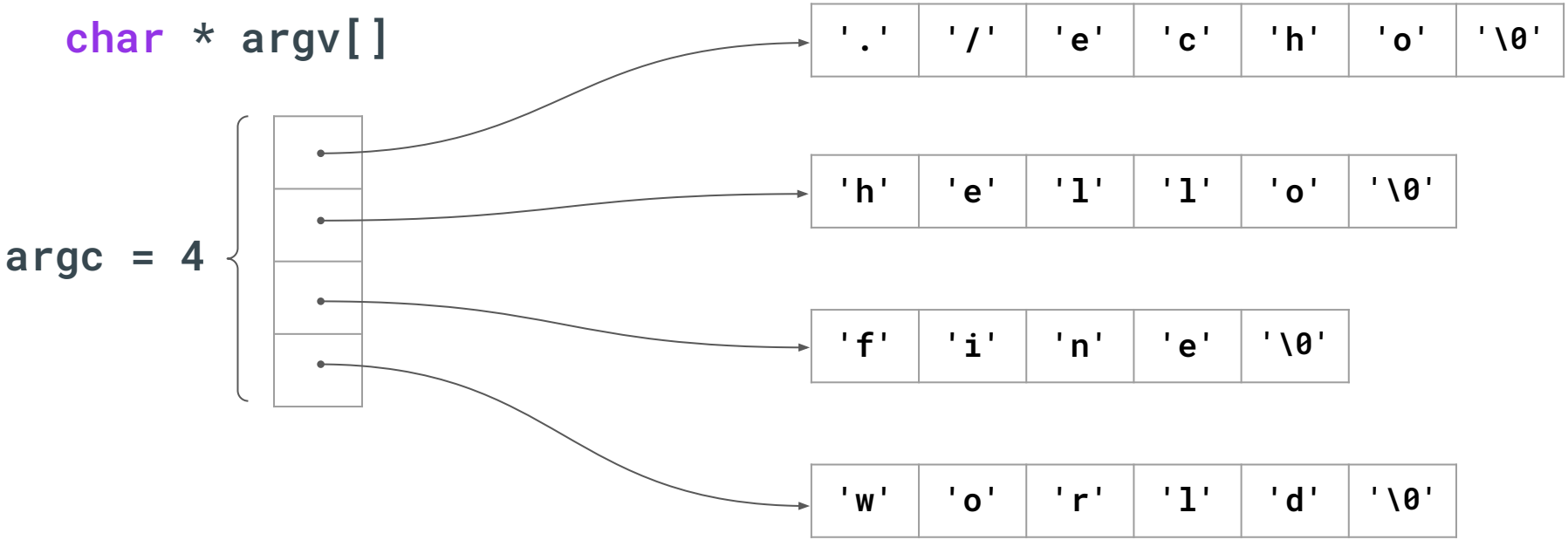
```c
#include <stdio.h>
int main(int argc, char * argv[]) {
  int i;
  for(i = 0 ; i < argc ; i++) {
    printf("%s\n", argv[i]);
  }
  return 0;
}
```

# Το περίφημο argv

```c
#include <stdio.h>
int main(int argc, char * argv[]) {
  int i;
  for(i = 0 ; i < argc ; i++) {
    printf("%s\n", argv[i]);
  }
  return 0;
}
```

```
$ ./echo hello fine world
./echo
hello
fine
world
```

# Το περίφημο `argv`



# Δυναμικοί Πίνακες με την συνάρτηση malloc()

Με την βοήθεια των δεικτών, μπορούμε να χρησιμοποιήσουμε δυναμικούς πίνακες, πίνακες των οποίων το μέγεθος καθορίζεται δυναμικά, δηλαδή την στιγμή που τρέχει το πρόγραμμα. Γενική μορφή:

```
τύπος * όνομα = malloc(μέγεθος * sizeof(τυπος));
```

Δήλωση δείκτη σε δεδομένα **τύπος (type)** - περιγράφει τον τύπο κάθε στοιχείου του πίνακα

Η συνάρτηση **malloc** επιστρέφει την διεύθυνση της μνήμης που θα βάλουμε τα στοιχεία του πίνακα

Ο αριθμός των **bytes** του πίνακα περιγράφει πόσος χώρος πρέπει να δεσμευτεί για να χωρέσει **μέγεθος** φορές τον τύπο

# Δυναμικοί Πίνακες με την συνάρτηση malloc()

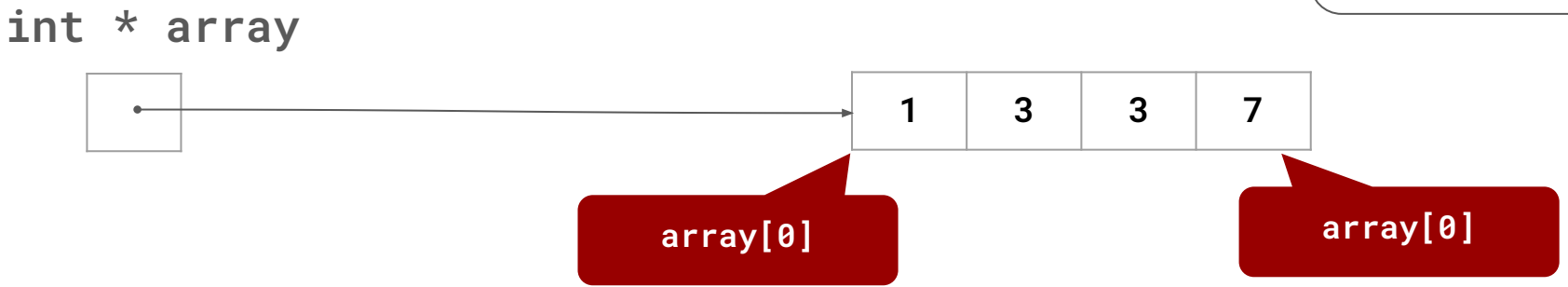
Με την βοήθεια των δεικτών, μπορούμε να χρησιμοποιήσουμε δυναμικούς πίνακες, πίνακες των οποίων το μέγεθος καθορίζεται δυναμικά, δηλαδή την στιγμή που τρέχει το πρόγραμμα. Παράδειγμα:

```
int * array = malloc(4 * sizeof(int));
```

stdlib.h

Δημιουργία πίνακα 4 ακεραίων

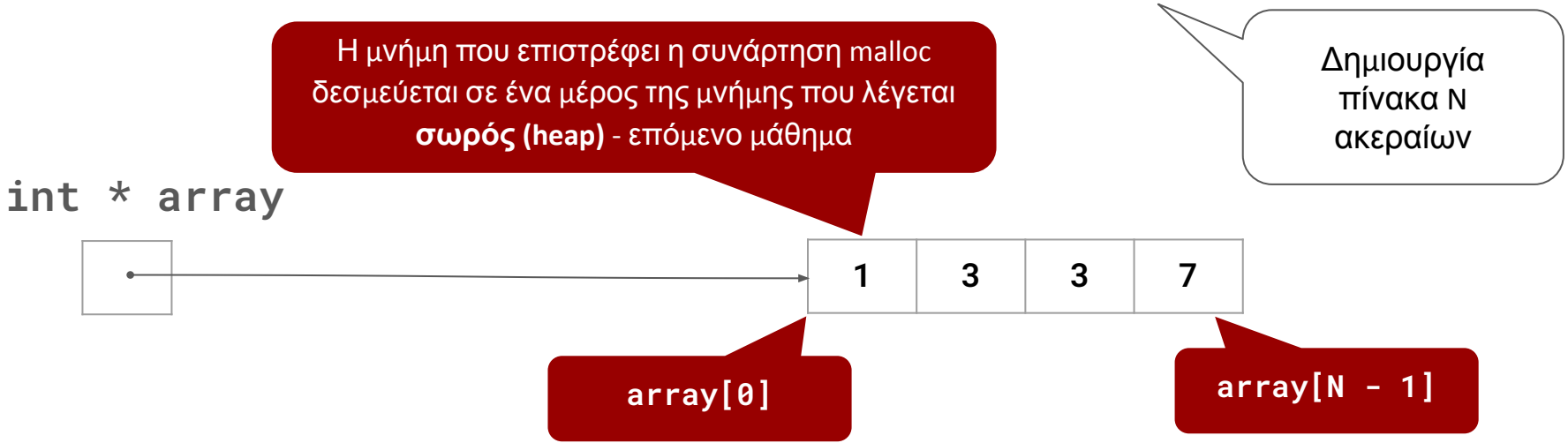
Η μνήμη μετά την εκτέλεση της malloc θα δείχνει ως εξής:



# Δυναμικοί Πίνακες με την συνάρτηση malloc()

Με την βοήθεια των δεικτών, μπορούμε να χρησιμοποιήσουμε δυναμικούς πίνακες, πίνακες των οποίων το μέγεθος καθορίζεται δυναμικά, δηλαδή την στιγμή που τρέχει το πρόγραμμα. Παράδειγμα:

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



# Δυναμικοί Πίνακες με την συνάρτηση malloc()

```
int * nums = malloc(100 * sizeof(int));

double * coeffs = malloc(100 * sizeof(double));

char * str = malloc(100 * sizeof(char));
```

Πόση μνήμη δεσμεύεται με καθεμιά από τις παραπάνω κλήσεις;

Τι θα τυπώσει το ακόλουθο:

```
printf("%d %d %d\n", sizeof(nums), sizeof(coeffs), sizeof(str));
```

# Δυναμικοί Πίνακες με την συνάρτηση malloc()

```
int * nums = malloc(100 * sizeof(int));

double * coeffs = malloc(100 * sizeof(double));

char * str = malloc(100 * sizeof(char));
```

Πόση μνήμη δεσμεύεται με καθεμιά από τις παραπάνω κλήσεις;

Τι θα τυπώσει το ακόλουθο:

```
printf("%d %d %d\n", sizeof(nums), sizeof(coeffs), sizeof(str));
```

```
$ ./dynamic
8 8 8
```

# Για την επόμενη φορά

- Καλύψαμε έννοιες από τις σελίδες 73-103 από τις σημειώσεις του κ. Σταματόπουλου - διαβάστε τις και εσείς με προσωπικό διάβασμα.

Ευχαριστώ και καλό σαβκο εύχομαι!
Keep Coding ;)