ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Τμήμα Πληροφορικής και Τηλεπικοινωνιών • Εισαγωγή στον Προγραμματισμό

Περιεχόμενα

Κεφάλαιο 12: Δείκτες και Πίνακες

> **Στόχοι**: μετά από αυτό το κεφάλαιο θα μπορείτε να χρησιμοποιείτε πολυδιάστατους πίνακες και να υπολογίζετε πού βρίσκεται στη μνήμη το `a[i][j]`· να χειρίζεστε δείκτες σε δείκτες, πίνακες από δείκτες και το `argv`· να δεσμεύετε δυναμικούς πίνακες με `malloc`· και να εξηγείτε τι είναι το endianness.
>
> **Προαπαιτούμενα**: Κεφάλαιο 10, Κεφάλαιο 11
>
> **Χρόνος μελέτης**: ~2,5 ώρες

Σύνοψη

Η διάλεξη πηγαίνει τους πίνακες και τους δείκτες ένα βήμα πιο πέρα. Οι δισδιάστατοι πίνακες της C είναι πίνακες από πίνακες, αποθηκευμένοι γραμμή-γραμμή σε συνεχόμενη μνήμη, οπότε η θέση του `a[i][j]` βγαίνει με έναν απλό τύπο. Οι δείκτες σε δείκτες και οι πίνακες από δείκτες

(με κορυφαίο παράδειγμα το `argv`) προσθέτουν ένα επίπεδο έμμεσης αναφοράς. Η `malloc` δίνει πίνακες με μέγεθος που αποφασίζεται κατά την εκτέλεση, και το endianness εξηγεί τη σειρά των bytes ενός ακεραίου. Όλα αυτά στηρίζουν τις δομές δεδομένων του δεύτερου μισού του μαθήματος.

Θεωρία

§12.1 Ο πίνακας στη μνήμη

Ένας **πίνακας (array)** κρατά δεδομένα ίδιου τύπου (Κεφάλαιο 10). Στη δήλωση `int bears[100];` ο **τύπος (type)** ορίζει πόση μνήμη παίρνει κάθε στοιχείο, το **όνομα (name)** κάνει τον μεταγλωττιστή να διαλέξει μια διεύθυνση για τον πίνακα, και το **μέγεθος (size)** πόσα στοιχεία θα κρατήσει. Το μέγεθος είναι στατικό: δεν αλλάζει κατά την εκτέλεση. Στα στοιχεία αναφερόμαστε με τη **θέση (index)** τους, `bears[0]` έως `bears[99]`, και αυτά κάθονται σε συνεχόμενες θέσεις. Με `sizeof(int) == 4` και αρχή στη διεύθυνση 4:

Bytes	0–3	4–7	8–11	…	400–403
Περιεχόμενο	(άλλο)	`bears[0]`	`bears[1]`	…	`bears[99]`

Ο πίνακας πιάνει $4 \cdot 100 = 400$ bytes (4–403), και το `bears[i]` βρίσκεται στη διεύθυνση «αρχή + `i * sizeof(int)`».

§12.2 Δισδιάστατοι πίνακες

Εικόνες, επιστημονικά και οικονομικά δεδομένα ή μια σκακιέρα έχουν φυσικά περισσότερες **διαστάσεις (dimensions)**. Ένας **δισδιάστατος πίνακας (two-dimensional array)** είναι πίνακας από (υπο)πίνακες: τύπος `όνομα[γραμμές][στήλες];`. Οι **γραμμές (rows)** είναι πόσους υποπίνακες έχει (1η διάσταση) και οι **στήλες (columns)** πόσα στοιχεία έχει ο καθένας (2η διάσταση), άρα συνολικά γραμμές × στήλες στοιχεία. Το `a[i][j]` είναι το στοιχείο της γραμμής `i` και της στήλης `j`, με μέτρηση από το 0, και χρησιμοποιείται όπως μια απλή μεταβλητή. Η αρχικοποίηση δίνει μία λίστα σε αγκύλες ανά γραμμή (εδώ η γραμμή 0 είναι 1, 4, 7, 10 και η γραμμή 1 είναι 3, 6, 9, 12):

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

Σε αρχικοποίηση μπορεί να παραλειφθεί μόνο το μέγεθος της **πρώτης** διάστασης (`char arr[][2] = {{'a','b'},{'c','d'}};`, από τις σημειώσεις)· ο λόγος φαίνεται στον υπολογισμό διευθύνσεων παρακάτω.

§12.3 Αποθήκευση κατά γραμμές και sizeof

Η μνήμη είναι μονοδιάστατη, γι' αυτό η C αποθηκεύει τον πίνακα **κατά γραμμές (row-major order)**: όλη η γραμμή 0, αμέσως μετά η γραμμή 1. Για τον παραπάνω `array`, με αρχή στο 100 (κάθε στοιχείο πιάνει 4 bytes):

Bytes	100	104	108	112	116	120	124	128
Τιμή	1	4	7	10	3	6	9	12
Στοιχείο	`[0][0]`	`[0][1]`	`[0][2]`	`[0][3]`	`[1][0]`	`[1][1]`	`[1][2]`	`[1][3]`

Αφού ο πίνακας είναι πίνακας από πίνακες, το `array[0]` είναι από μόνο του ένας πίνακας 4 ακεραίων (bytes 100–115) και το `array[1]` ο επόμενος (116–131). Γι' αυτό `sizeof(array[0])` και `sizeof(array[1])` τυπώνουν 16 και `sizeof(array)` 32 (οι διαφάνειες τυπώνουν με `%d`· για `size_t` το σωστό είναι `%zu`), και `sizeof(array) / sizeof(array[0])` δίνει το πλήθος των γραμμών.

§12.4 Υπολογισμός διεύθυνσης στοιχείου

Για `int array[X][Y];`, για να φτάσουμε στο `a[i][j]` προσπερνάμε `i` γραμμές των `Y` ακεραίων και μετά `j` ακεραίους:

```
&a[i][j] = StartAddressOfArray + i * Y * sizeof(int) + j * sizeof(int)
IndexOfElementIJ = i * Y + j
```

Ο δεύτερος είναι η θέση σε έναν νοητό μονοδιάστατο πίνακα `X * Y` ακεραίων. Και οι δύο χρειάζονται **μόνο τον αριθμό `Y` των στηλών**. Γι' αυτό η πρώτη διάσταση μπορεί να παραλειφθεί, και μια συνάρτηση που δέχεται δισδιάστατο πίνακα γράφει την παράμετρο π.χ. `int matr[][12]` (σημειώσεις, «Πολυδιάστατοι πίνακες»). Έλεγχος με τον πίνακα παραπάνω: το `a[1][2]` είναι στο $100 + 1 \cdot 4 \cdot 4 + 2 \cdot 4 = 124$, όπου πράγματι βρίσκεται το 9.

§12.5 Πίνακες περισσότερων διαστάσεων

Πίνακες τριών ή περισσότερων διαστάσεων λειτουργούν με τον ίδιο τρόπο, με περισσότερα ζεύγη αγκυλών (π.χ. `int rubiksCube[6][3][3];`), και είναι πάλι συνεχόμενοι στη μνήμη: για `[X][Y][Z]` η θέση του `[i][j][k]` είναι `i * Y * Z + j * Z + k`. Η διάλεξη ρωτά: **είναι απαραίτητοι οι πολυδιάστατοι πίνακες;** Όχι· μπορούμε να κρατήσουμε τα ίδια δεδομένα σε μονοδιάστατο πίνακα και να υπολογίζουμε μόνοι μας τη θέση. Είναι όμως βολικοί, γιατί τον λογαριασμό τον κάνει ο μεταγλωττιστής.

§12.6 Δείκτες: υπενθύμιση

Ένας **δείκτης (pointer)** κρατά μια διεύθυνση μνήμης (Κεφάλαιο 11). Ο μοναδιαίος `*` κάνει **αποαναφορά (dereference)**: μετά το `int *pointer = &x;` το `*pointer` είναι ισοδύναμο με

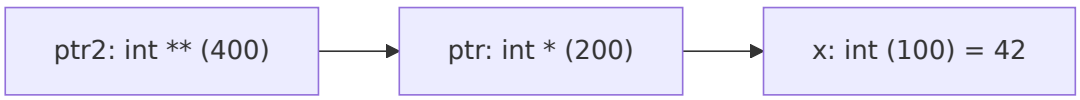
το x, για ανάγνωση και για εγγραφή (αν το x είναι στη διεύθυνση 100, ο pointer περιέχει την τιμή 100).

Στην **αριθμητική δεικτών (pointer arithmetic)**, αν ο p δείχνει σε int, ο p + 2 δείχνει δύο ακεραίους (όχι bytes) πιο μετά, το *(p + 2) ισοδυναμεί με p[2] και το p++ πάει στο επόμενο στοιχείο. Το όνομα ενός πίνακα σε έκφραση μετατρέπεται σε δείκτη στο πρώτο του στοιχείο, άρα το p = x; κάνει τον p να δείχνει στο x[0].

§12.7 Δείκτης σε δείκτη

Ένας δείκτης έχει κι αυτός διεύθυνση, που μπορεί να αποθηκευτεί σε άλλον δείκτη. Ο **δείκτης σε δείκτη (pointer to pointer)** δηλώνεται με δύο αστερίσκους: μετά από int x = 42; int *ptr = &x; int **ptr2 = &ptr; ο ptr2 είναι δείκτης σε int * (διαβάστε τον τύπο από δεξιά). Με τις διευθύνσεις της διαφάνειας:

Μεταβλητή	Τύπος	Διεύθυνση	Τιμή
x	int	100	42
ptr	int *	200	100
ptr2	int **	400	200



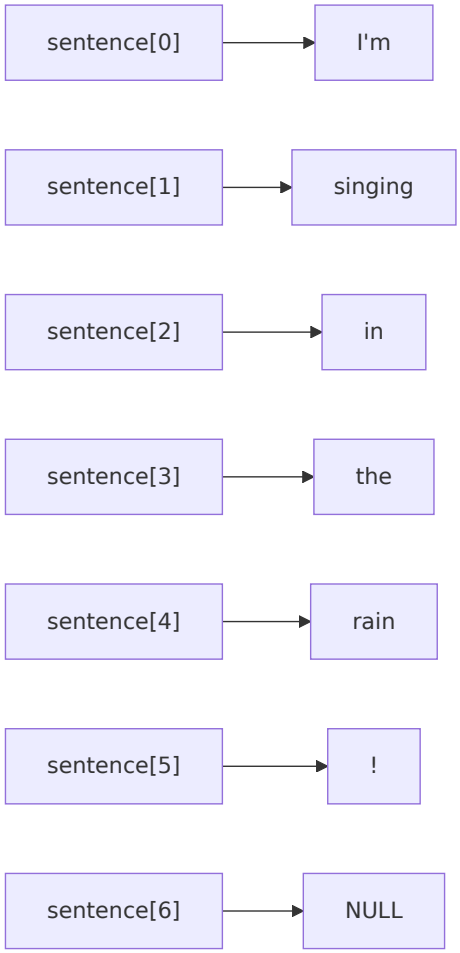
*Σχήμα: κάθε * ακολουθεί ένα βέλος· *ptr2 είναι ο ptr (100), **ptr2 ο x (42).*

Η διάλεξη ρωτά τι τυπώνει το printf("%p %d", *ptr2, **ptr2);: μια διεύθυνση (σε δεκαεξαδικό, διαφορετική σε κάθε εκτέλεση) και το 42. Το ίδιο γενικεύεται σε int ***. **Γιατί έχει σημασία**; Δέντρα και γράφοι είναι κόμβοι που δείχνουν σε άλλους κόμβους (Κεφάλαιο 21), και το argv και οι δυναμικοί δισδιάστατοι πίνακες έχουν τύπους char ** και int **.

§12.8 Πίνακες από δείκτες

Οι δείκτες μπαίνουν και σε πίνακες: ο **πίνακας από δείκτες (array of pointers)** int *ptr[3]; έχει τρία στοιχεία τύπου int *. Το *ptr[2] σημαίνει *(ptr[2]), γιατί οι αγκύλες έχουν μεγαλύτερη προτεραιότητα από τον μοναδιαίο *.

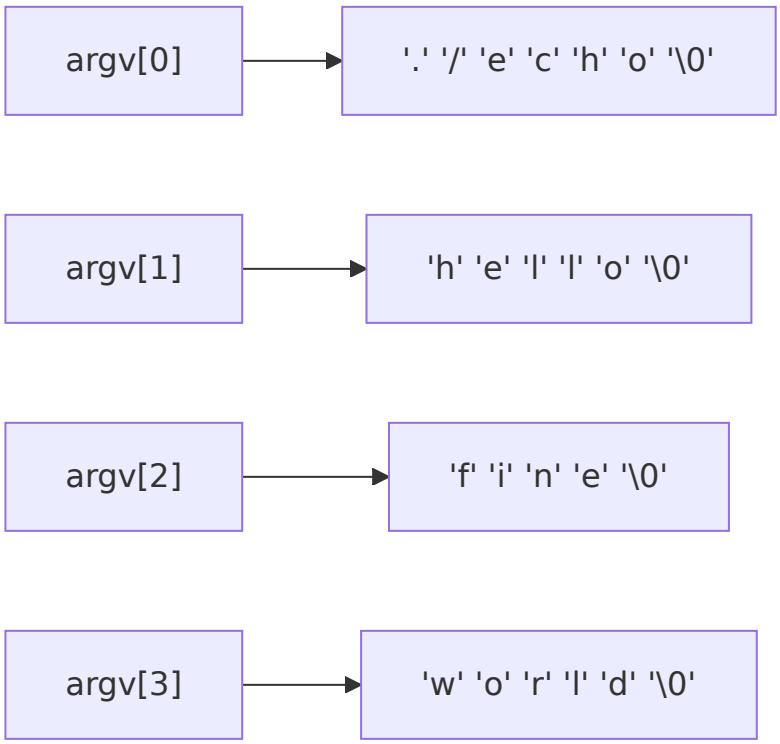
Η συνηθέστερη χρήση είναι ένας πίνακας συμβολοσειρών, char *sentence[]: κάθε στοιχείο δείχνει στον πρώτο χαρακτήρα μιας συμβολοσειράς, και σε αντίθεση με έναν δισδιάστατο πίνακα char οι συμβολοσειρές μπορούν να έχουν διαφορετικά μήκη. Συχνά το τελευταίο στοιχείο είναι ο **κενός δείκτης (null pointer)** NULL, που λειτουργεί ως «φρουρός»: ένας βρόχος με συνθήκη sentence[i] σταματά εκεί χωρίς να ξέρει το πλήθος.



Σχήμα: πίνακας από δείκτες σε συμβολοσειρές διαφορετικού μήκους, με `NULL` *στο τέλος.*

§12.9 Το argv

Το πιο γνωστό παράδειγμα πίνακα από δείκτες είναι η παράμετρος της `main` για τα **ορίσματα γραμμής εντολών (command-line arguments)**: `int main(int argc, char * argv[])`. Το `argc` είναι το πλήθος των ορισμάτων **μαζί με το όνομα του προγράμματος**, και το `argv[i]` δείχνει στη συμβολοσειρά του `i`-οστού ορίσματος. Για `./echo hello fine world` έχουμε `argc = 4`:



Σχήμα: το `argv` *για* `./echo hello fine world` *(*`argc = 4`*).*

Τα πραγματικά ορίσματα ξεκινούν από το `argv[1]`, και το `argv[argc]` είναι `NULL` (σημειώσεις, «Ορίσματα γραμμής εντολών»). Επειδή ένας πίνακας περνά σε συνάρτηση ως δείκτης στο πρώτο του στοιχείο, η παράμετρος γράφεται ισοδύναμα `char **argv`.

§12.10 Δυναμικοί πίνακες με malloc

Όταν το μέγεθος γίνεται γνωστό μόνο κατά την εκτέλεση (π.χ. διαβάζεται από την είσοδο), χρησιμοποιούμε **δυναμικούς πίνακες (dynamic arrays)**:

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

Η `malloc` (από το `stdlib.h`) δεσμεύει μνήμη και επιστρέφει τη διεύθυνσή της, που την κρατά ένας δείκτης στον τύπο των στοιχείων. Το όρισμα είναι σε **bytes**, γι' αυτό πολλαπλασιάζουμε το πλήθος με `sizeof(τύπος)`. Το `int * array = malloc(N * sizeof(int));` δημιουργεί πίνακα `N` ακεραίων, που χάρη στην αριθμητική δεικτών χρησιμοποιείται κανονικά, από `array[0]` έως `array[N - 1]`. Η μνήμη αυτή βρίσκεται στον **σωρό (heap)**, το θέμα του Κεφαλαίου 13.

Από τις σημειώσεις: η `malloc` δεν αρχικοποιεί τη μνήμη (οι τιμές 1, 3, 3, 7 της διαφάνειας είναι ενδεικτικές)· αν αποτύχει επιστρέφει `NULL`, οπότε ελέγχουμε πάντα το αποτέλεσμα· και ό,τι δεσμεύσαμε το αποδεσμεύουμε με `free`. Το `sizeof` ενός δείκτη είναι το μέγεθος μιας διεύθυνσης (8 bytes σε σύστημα 64 bit), όχι του μπλοκ, οπότε το πλήθος των στοιχείων το κρατάμε σε

μεταβλητή. Ένας δυναμικός δισδιάστατος πίνακας `N × M` φτιάχνεται με έναν `int **` που δείχνει σε `N` δείκτες γραμμών, με κάθε γραμμή από δική της `malloc`, ή με έναν μονοδιάστατο πίνακα `N * M` και τη θέση `i * M + j`.

§12.11 Endianness

Ένας `int` αποτελείται από πολλά bytes. **Endianness** λέμε τη σειρά με την οποία αποθηκεύονται: **little endian** από το μικρότερο (λιγότερο σημαντικό) byte προς το μεγαλύτερο, **big endian** αντίστροφα. Για τα διαδοχικά bytes `68 65 6c 6c` (ASCII `'h' 'e' 'l' 'l'`):

Μηχάνημα	byte +0	byte +1	byte +2	byte +3	Ακέραιος
little endian	`68`	`65`	`6c`	`6c`	`0x6c6c6568`
big endian	`68`	`65`	`6c`	`6c`	`0x68656c6c`

Οι x86 και οι περισσότεροι ARM είναι little endian (τα σχήματα μνήμης των διαφανειών για τους δείκτες είναι σχεδιασμένα big endian, που διαβάζεται ευκολότερα). Το endianness μετράει σε δυαδικά δεδομένα, π.χ. κεφαλίδες αρχείων εικόνας και ήχου.

Παραδείγματα

§12.12 Η σκακιέρα

Εφαρμόζει τους «Δισδιάστατους πίνακες»: κάθε κελί είναι ένας `char` (κεφαλαία για τον έναν παίκτη, πεζά για τον άλλο, κενό για άδειο τετράγωνο).

```c
#include <stdio.h>

int main() {
    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';
    return 0;
}
```

Οι δύο αναθέσεις κάνουν την κίνηση e2–e4 (γραμμή 1 → γραμμή 3, στήλη 4). Με τη σύμβαση του πίνακα το πιόνι θα έπρεπε να είναι `'P'`· η διαφάνεια γράφει `'p'`. Για εκτύπωση, ο εξωτερικός βρόχος διατρέχει τις γραμμές και ο εσωτερικός τις στήλες.

§12.13 Ο κύβος του Rubik

Εφαρμόζει τους «Πίνακες περισσότερων διαστάσεων»: 6 έδρες × 3 × 3 = 54 ακέραιοι.

```
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)
};
```

§12.14 Βρόχος με αριθμητική δεικτών

Εφαρμόζει τη «Δείκτες: υπενθύμιση». 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++;
```

Η συνθήκη είναι **ανάθεση**: αντιγράφει στο *p την τιμή δύο θέσεις πιο μετά, και αφού η τιμή μιας ανάθεσης είναι η τιμή που ανατέθηκε, ο βρόχος σταματά μόλις αντιγραφεί ένα 0. Ο πίνακας αλλάζει όσο τρέχει ο βρόχος, οπότε ιχνηλατήστε βήμα-βήμα. Για όσους δυσκολεύονται με τους δείκτες, η διάλεξη προτείνει το βίντεο των βοηθών (δείτε το Διάβασμα).

§12.15 Πίνακας από δείκτες σε ακεραίους

Εφαρμόζει τους «Πίνακες από δείκτες»:

```
#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
```

§12.16 Μια πρόταση ως πίνακας από συμβολοσειρές

Εφαρμόζει τους «Πίνακες από δείκτες» με φρουρό `NULL`:

```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
!
```

§12.17 Ένα απλό echo

Εφαρμόζει «Το argv»: τυπώνει όλα τα ορίσματα, μαζί με το όνομα του προγράμματος. Για ορίσματα-αριθμούς (π.χ. `argcalc.c` του εργαστηρίου 8) χρειάζεται η `atoi`.

```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
```

§12.18 Πόση μνήμη δεσμεύει η malloc

Εφαρμόζει τους «Δυναμικούς πίνακες». Πόση μνήμη δεσμεύει κάθε κλήση και τι τυπώνει το `printf`;

```
int * nums = malloc(100 * sizeof(int));
double * coeffs = malloc(100 * sizeof(double));
char * str = malloc(100 * sizeof(char));
printf("%zu %zu %zu\n", sizeof(nums), sizeof(coeffs), sizeof(str));
```

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

Κάθε κλήση δεσμεύει `100 * sizeof(τύπος)` bytes, αλλά το `printf` τυπώνει το μέγεθος των ίδιων των δεικτών: 8 bytes ο καθένας σε σύστημα 64 bit. Για την άσκηση `array.c` του εργαστηρίου 7: διαβάστε το `N`, δεσμεύστε, ελέγξτε για `NULL`, και στο τέλος `free`.

§12.19 Βλέποντας τα bytes ενός ακεραίου

Εφαρμόζει το «Endianness». Μετατρέπουμε (cast) τη διεύθυνση του `x` σε `char *`, που προχωρά ένα byte τη φορά. Τι θα τυπώσει;

```
#include <stdio.h>
int main() {
  int x = 42;
  char * bytes = (char*)&x;
  int i;
  for(i = 0; i < sizeof(int) / sizeof(char); i++)
     printf("%02x\n", bytes[i]);
  return 0;
}
```

```
$ ./int
2a
00
00
00
```

Το 42 είναι `0x0000002a`· πρώτο τυπώνεται το λιγότερο σημαντικό byte, άρα το μηχάνημα είναι little endian. Επειδή το `char` είναι συνήθως προσημασμένο, ένα byte `0x80` θα έβγαινε `ffffff80` με `%02x`· με `unsigned char *` αυτό δεν συμβαίνει.

Κύρια σημεία

1. Ένας δισδιάστατος πίνακας είναι πίνακας από πίνακες, τύπος `όνομα[γραμμές][στήλες]`, αρχικοποιείται με μία λίστα ανά γραμμή, και το `a[i][j]` είναι μια απλή μεταβλητή.

2. Αποθηκεύεται κατά γραμμές σε συνεχόμενη μνήμη· `sizeof(array[0])` είναι μία γραμμή, `sizeof(array)` όλος ο πίνακας.
3. Για `int array[X][Y]` η θέση του `a[i][j]` είναι `i * Y + j`: χρειάζεται μόνο το `Y`, γι' αυτό μόνο η πρώτη διάσταση μπορεί να παραλειφθεί.
4. Πίνακες περισσότερων διαστάσεων λειτουργούν ίδια· δεν είναι απαραίτητοι, αλλά απλοποιούν τον κώδικα.
5. Το `*pointer` ισοδυναμεί με τη μεταβλητή όπου δείχνει ο δείκτης· ένας `int **` κρατά τη διεύθυνση ενός δείκτη, και κάθε `*` ακολουθεί ένα επίπεδο.
6. Οι δείκτες μπαίνουν σε πίνακες· ένας πίνακας `char *` με `NULL` στο τέλος κρατά συμβολοσειρές διαφορετικού μήκους.
7. Το `argv` είναι πίνακας από `char *` (ισοδύναμα `char **`) με `argc` στοιχεία, και το `argv[0]` είναι το όνομα του προγράμματος.
8. Το `malloc(μέγεθος * sizeof(τύπος))` φτιάχνει στον σωρό πίνακα με μέγεθος που αποφασίζεται κατά την εκτέλεση· το `sizeof` του δείκτη είναι 8, όχι το μέγεθος του μπλοκ.
9. Endianness είναι η σειρά των bytes ενός ακεραίου: στο little endian πρώτο είναι το λιγότερο σημαντικό· με έναν `char *` το διαπιστώνουμε.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
πίνακας / θέση	array / index	Στοιχεία ίδιου τύπου σε συνεχόμενη μνήμη / ο `i` στο `a[i]`.
δισδιάστατος πίνακας	two-dimensional array	Πίνακας από πίνακες, `a[γραμμές][στήλες]`.
αποθήκευση κατά γραμμές	row-major order	Οι γραμμές αποθηκεύονται η μία μετά την άλλη.
αποαναφορά	dereference	Πρόσβαση με `*` εκεί όπου δείχνει ένας δείκτης.
δείκτης σε δείκτη	pointer to pointer	Δείκτης που κρατά διεύθυνση δείκτη, π.χ. `int **`.
πίνακας από δείκτες	array of pointers	Πίνακας με στοιχεία τύπου δείκτη, π.χ. `char *s[5]`.
κενός δείκτης	null pointer (`NULL`)	Δείκτης που δεν δείχνει πουθενά· συχνά σημαδεύει το τέλος.
ορίσματα γραμμής εντολών	command-line arguments	Οι λέξεις της κλήσης, στα `argc`/`argv`.
δυναμικός πίνακας	dynamic array	Πίνακας με μέγεθος που αποφασίζεται κατά την εκτέλεση.
σωρός	heap	Η περιοχή μνήμης από την οποία δεσμεύει η `malloc`.

σειρά bytes	endianness	Η σειρά αποθήκευσης των bytes ενός ακεραίου.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 12, σελ. 1–45: μονοδιάστατοι πίνακες 5–10· δισδιάστατοι 11–22· πολυδιάστατοι 23· δείκτες σε δείκτες 24–28· πίνακες από δείκτες και `argv` 29–35· `malloc` 36–40· endianness 41–43.
- **Σημειώσεις**: η διάλεξη ζητά να διαβάσετε τις σελ. 73–103:
  - Κεφάλαιο 5: «Δείκτες» (K04, σελ. 72–77), «Περί ανάγνωσης ακεραίων (και όχι μόνο)» (78–79), «Πίνακες» (80–85), «Ιστόγραμμα συχνοτήτων γραμμάτων στην είσοδο» (86–87).
  - Κεφάλαιο 6: «Δυναμική δέσμευση μνήμης» (K04, σελ. 88–92), «Συμβολοσειρές» (93–96), «Πίνακες δεικτών και δείκτες σε δείκτες» (97), «Ορίσματα γραμμής εντολών» (98–99), «Πολυδιάστατοι πίνακες» (100–102), «Αρχικοποίηση πινάκων» (103).
- **Εργαστήριο**: Εργαστήριο 6: `pointers.c`· Εργαστήριο 7: `twodim.c`, `array.c`, `mines.c`· Εργαστήριο 8: `argcalc.c`.
- **Άλλα**: Endianness (Wikipedia)· βίντεο «Εισαγωγή στους Pointers» από τους βοηθούς του μαθήματος· `man 3 malloc`.

Συχνά λάθη

- `a[i, j]` **αντί για** `a[i][j]`. Το κόμμα είναι τελεστής, άρα σημαίνει `a[j]`.
- **Μπέρδεμα γραμμών και στηλών**. Στο `int a[2][4]` το `a[3][1]` είναι εκτός ορίων (`Segmentation fault`): ο πρώτος δείκτης είναι η γραμμή.
- **Παράλειψη της δεύτερης διάστασης**. Το `int m[][] = {{1, 2}, {3, 4}};` δίνει `array type has incomplete element type`. Γράψτε `int m[][2]`.
- `sizeof` **δείκτη ως μέγεθος πίνακα** (8, όχι 400) ή `sizeof` **με** `%d` (`format '%d' expects argument of type 'int'`): κρατήστε το πλήθος σε μεταβλητή και τυπώνετε το `sizeof` με `%zu`.
- **Ξεχασμένο** `sizeof` **ή** `stdlib.h` **στη** `malloc`. Το `malloc(100)` χωρά μόνο 25 `int`· χωρίς `#include <stdlib.h>` ο gcc λέει `implicit declaration of function 'malloc'`.
- **Χωρίς έλεγχο για** `NULL` **ή αρχικοποίηση**. Η `malloc` μπορεί να αποτύχει και δεν μηδενίζει τη μνήμη: ελέγξτε `if (array == NULL)` και γράψτε πριν διαβάσετε.
- `argv[1]` **χωρίς έλεγχο του** `argc`. Χωρίς όρισμα το `argv[1]` είναι `NULL` και το `atoi(argv[1])` δίνει `Segmentation fault`. Ελέγξτε `if (argc < 2)`· και θυμηθείτε ότι για `./prog a b c` το `argc` είναι 4.
- **Πίνακας από δείκτες χωρίς φρουρό** `NULL`. Το `for (i = 0; sentence[i]; i++)` βγαίνει έξω από τον πίνακα.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- K12.9 Διεύθυνση στοιχείου δισδιάστατου πίνακα (33% σωστές): Το 29% επέλεξε `155`: μέτρησαν σωστά ότι προηγούνται 55 στοιχεία, αλλά ξέχασαν να τα πολλαπλασιάσουν με `sizeof(int)` για να τα κάνουν bytes.
- K12.8 Χαρακτήρας από πίνακα συμβολοσειρών (38% σωστές): Το 23% επέλεξε `"fine"` και άλλο 22% `'e'`: οι πρώτοι σταμάτησαν στο `strings[1]` αγνοώντας τον δεύτερο δείκτη, οι δεύτεροι μέτρησαν τις θέσεις από το 1.
- K12.7 sizeof μιας γραμμής (42% σωστές): Το 26% επέλεξε `1`, θεωρώντας ότι το `map[5]` είναι ένας χαρακτήρας. Στην πραγματικότητα το `map[5]` είναι ολόκληρη η 6η γραμμή, ένας πίνακας 10 `char`.
- K12.6 Ο τύπος του argv (44% σωστές): Το 34% επέλεξε «Ένας δείκτης σε πίνακες από χαρακτήρες», διαβάζοντας τη δήλωση ανάποδα. Οι αγκύλες δένουν πιο ισχυρά από το `*`, άρα το `argv` είναι πρώτα πίνακας, και τα στοιχεία του είναι `char *`.

Ερωτήσεις κατανόησης

- E12.1 Πόσα bytes έχει ο `double m[3][5];` (`sizeof(double) == 8`), και πού βρίσκεται το `a[2][3]` του `int a[10][20];` αν αυτός ξεκινά στο 1000;[1]
- E12.2 Γιατί ο αριθμός των γραμμών δεν εμφανίζεται στον τύπο της διεύθυνσης του `a[i][j]`;[2]
- E12.3 Για `./prog one two`, ποιο είναι το `argc` και τι περιέχει το `argv[0]`;[3]
- E12.4 Τι δεσμεύει το `malloc(100 * sizeof(double))` και πόσο είναι το `sizeof` του δείκτη που το κρατά;[4]
- E12.5 Με ποια σειρά βρίσκονται τα bytes του `0x12345678` σε little endian;[5]

Kahoot από το αμφιθέατρο (K12.1–K12.9)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- K12.1 Διαστάσεις πίνακα: 93% σωστές απαντήσεις
- K12.2 Στατικοί και δυναμικοί πίνακες: 92% σωστές απαντήσεις
- K12.3 Πλήθος στοιχείων δισδιάστατου πίνακα: 81% σωστές απαντήσεις
- K12.4 sizeof δισδιάστατου πίνακα: 68% σωστές απαντήσεις
- K12.5 Το τελευταίο στοιχείο δισδιάστατου πίνακα: 67% σωστές απαντήσεις
- K12.6 Ο τύπος του argv: 44% σωστές απαντήσεις
- K12.7 sizeof μιας γραμμής: 42% σωστές απαντήσεις

[1] 120 bytes· $1000 + 2 \cdot 20 \cdot 4 + 3 \cdot 4 = 1172$.

[2] Για να φτάσουμε στη γραμμή `i` προσπερνάμε `i` γραμμές μήκους `Y`.

[3] `argc` είναι 3 και το `argv[0]` δείχνει στο `"./prog"`.

[4] 800 bytes στον σωρό· ο δείκτης είναι 8 bytes σε σύστημα 64 bit.

[5] `78`, `56`, `34`, `12`.

- K12.8 Χαρακτήρας από πίνακα συμβολοσειρών: 38% σωστές απαντήσεις
- K12.9 Διεύθυνση στοιχείου δισδιάστατου πίνακα: 33% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A12.1–A12.7)

- A12.1 Στοιχείο δισδιάστατου πίνακα: Διάλεξη 12, διαφάνεια 14 · ★☆☆ · trace · `slides-lec12-2d-element`
- A12.2 Πίνακας από δείκτες: Διάλεξη 12, διαφάνεια 29 · ★☆☆ · trace · `slides-lec12-array-of-pointers`
- A12.3 Δείκτης σε δείκτη: Διάλεξη 12, διαφάνεια 27 · ★☆☆ · trace · `slides-lec12-pointer-to-pointer`
- A12.4 Παράδειγμα endianness: Διάλεξη 12, διαφάνεια 42 · ★★☆ · trace · `slides-lec12-endianness`
- A12.5 Πόση μνήμη δεσμεύει η malloc και τι λέει το sizeof: Διάλεξη 12, διαφάνειες 39–40 · ★★☆ · short-answer · `slides-lec12-malloc-sizeof`
- A12.6 Είναι απαραίτητοι οι πολυδιάστατοι πίνακες;: Διάλεξη 12, διαφάνεια 23 · ★★☆ · short-answer · `slides-lec12-multidim-necessary`
- A12.7 Περιεχόμενα του x μετά από βρόχο με δείκτη: Διάλεξη 12, διαφάνεια 25 · ★★☆ · trace · `slides-lec12-pointer-copy-loop`

Εργαστήριο (A12.8–A12.11)

- A12.8 Δυναμική δέσμευση μνήμης για μονοδιάστατο πίνακα: Εργαστήριο 7, Άσκηση 2 · ★☆☆ · programming · `lab-lab07-array`
- A12.9 Ορίσματα γραμμής εντολής: Εργαστήριο 8, Άσκηση 2 · ★☆☆ · programming · `lab-lab08-argcalc`
- A12.10 Δισδιάστατοι πίνακες: Εργαστήριο 7, Άσκηση 1 · ★★☆ · programming · `lab-lab07-twodim`
- A12.11 Πετυχαίνοντας τον στόχο (Παλιό θέμα): Εργαστήριο 8, Άσκηση 3 · ★★☆ · programming · `lab-lab08-legolas`

Εργασίες (A12.12)

- A12.12 FauxtoShop: περιστροφή εικόνας BMP: Εργασία 2 (2023-24), Άσκηση 1 · ★★★ · programming · `hw-2023-hw2-fauxtoshop`

Θέματα εξετάσεων (A12.13–A12.22)

- A12.13 Αλλαγή Τέρματος: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex13-q2`
- A12.14 Τουρνουά: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex2-q2`

- **A12.15** Μέση Τιμή Τυχαίων Μεταβλητών - mean: Εξέταση Σεπτεμβρίου 2025, Θέμα 3 · ★☆☆ · programming · `exam-2025-sep-q3`
- **A12.16** Περιστροφή Πίνακα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex1-q3`
- **A12.17** Κινήσεις σε Πλέγμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex11-q3`
- **A12.18** Πολλαπλασιασμός Πινάκων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex8-q3`
- **A12.19** Πολύτιμοι Πίνακες: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex9-q3`
- **A12.20** Στατιστικές: Εξέταση Ιουλίου 2024, Θέμα 2 · ★★☆ · programming · `exam-2024-jul-q2`
- **A12.21** Κινούμενος Μέσος Όρος - sma: Εξέταση Ιανουαρίου 2025, Θέμα 3 · ★★☆ · programming · `exam-2025-jan-q3`
- **A12.22** Το μεγαλύτερο άλμα - polevault: Εξέταση Σεπτεμβρίου 2026, Θέμα 3 · ★★☆ · programming · `exam-2026-sep-q3`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A4.13** Άρτια Bits: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex8-q1`
- **A6.24** Πετυχαίνοντας τον Στόχο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex9-q2`
- **A9.17** Επεξεργασία Ήχου (soundwave): Εργασία 1 (2025-26), Άσκηση 1 · ★★★ · programming · `hw-2025-hw1-soundwave`
- **A10.20** Τοποθέτηση του Pacman: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex11-q2`
- **A10.21** Φορτωμένο Έλκηθρο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex3-q2`
- **A10.23** Μαγικό Ζευγάρι: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex5-q2`
- **A11.19** Ο Αλγόριθμος του Ευκλείδη (gcd): Εργασία 1 (2024-25), Άσκηση 1 · ★★☆ · programming · `hw-2024-hw1-gcd`
- **A11.21** Ο Αλγόριθμος RSA (rsa): Εργασία 1 (2024-25), Άσκηση 2 · ★★★ · programming · `hw-2024-hw1-rsa`
- **A11.17** Πίνακες και αριθμητική δεικτών: Εργαστήριο 6, Άσκηση 3 · ★★☆ · trace · `lab-lab06-pointers`
- **A13.8** Δυναμική δέσμευση μνήμης για δισδιάστατο πίνακα: Εργαστήριο 7, Άσκηση 3 · ★★☆ · programming · `lab-lab07-mines`
- **A13.9** Κινήσεις σε πλέγμα (Παλιό θέμα): Εργαστήριο 7, Άσκηση 5 · ★★★ · programming · `lab-lab07-pacman`
- **A13.4** Δυναμικός δισδιάστατος πίνακας MxN: Διάλεξη 13, διαφάνειες 56–58 · ★★☆ · programming · `slides-lec13-dynamic-2d`

- A14.13 Ταιριαστές Καρδιές: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex0-q2`
- A14.14 Εντοπισμός Διπλών Ορισμάτων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex1-q2`
- A14.22 Μεταμορφώσιμες Προτάσεις: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex5-q4`
- A14.12 Η συνάρτηση transform: Εξέταση Σεπτεμβρίου 2025, Θέμα 2 · ★☆☆ · trace · `exam-2025-sep-q2`
- A14.9 Επεξεργασία συμβολοσειρών: Εργαστήριο 8, Άσκηση 1 · ★★☆ · programming · `lab-lab08-string`
- A14.1 Είναι το πρώτο όρισμα ”--boo”;: Διάλεξη 14, διαφάνεια 42 · ★☆☆ · programming · `slides-lec14-check-boo`
- A15.2 Πολυπλοκότητα εύρεσης μέγιστου σε πίνακα N x N: Διάλεξη 15, διαφάνεια 23 · ★☆☆ · short-answer · `slides-lec15-complexity-find-max-2d`
- A15.4 Πολυπλοκότητα δυναμικού πίνακα με malloc: Διάλεξη 15, διαφάνεια 19 · ★☆☆ · short-answer · `slides-lec15-complexity-malloc`
- A16.10 char *array[], char **array και char array[10][10]: Διάλεξη 16, διαφάνεια 19 · ★★☆ · short-answer · `slides-lec16-char-pointer-arrays`
- A16.6 yes αν ένα στοιχείο υπάρχει σε δισδιάστατο πίνακα: Διάλεξη 16, διαφάνεια 24 · ★☆☆ · programming · `slides-lec16-search-2d`
- A17.11 Πλησιάζοντας στον Στόχο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex10-q4`
- A17.13 Η συνάρτηση compute: Εξέταση Ιανουαρίου 2026, Θέμα 2 · ★★☆ · trace · `exam-2026-jan-q2`
- A18.17 Αλλαγή Μεγέθους: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex4-q4`
- A18.13 Κόψιμο Αρχείων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 4 · ★☆☆ · programming · `exam-2023-fall-ex7-q4`
- A18.11 Προβλέποντας το Μέλλον (future): Εργασία 2 (2024-25), Άσκηση 1 · ★★☆ · programming · `hw-2024-hw2-future`
- A18.7 Μέτρηση στατιστικών αρχείων: Εργαστήριο 10, Άσκηση 4 · ★☆☆ · programming · `lab-lab10-count`
- A18.8 Σύγκριση αρχείων: Εργαστήριο 10, Άσκηση 3 · ★☆☆ · programming · `lab-lab10-filediff`
- A18.10 Αρχεία κειμένου: Εργαστήριο 10, Άσκηση 1 · ★★☆ · programming · `lab-lab10-more`
- A18.1 Μέγεθος δισδιάστατου πίνακα και μιας γραμμής του: Διάλεξη 18, διαφάνεια 2 · ★☆☆ · short-answer · `slides-lec18-2d-sizeof`
- A18.5 Ακέραιοι από αρχείο κειμένου με fread: Διάλεξη 18, διαφάνειες 50–51 · ★★☆ · trace · `slides-lec18-fread-int`
- A18.6 Μια λέξη σε char[7] με scanf: Διάλεξη 18, διαφάνειες 31–35 · ★★☆ · debug · `slides-lec18-scanf-string`

- A22.15 Zoomba: συντομότερη διαδρομή σε δωμάτιο: Εργασία 3 (2023-24), Άσκηση 1 · ★★★ · programming · `hw-2023-hw3-zoomba`
- A25.12 Λύσε τον Λαβύρινθο: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 4 · ★★★ · programming · `exam-2024-dec-q4`
- A25.14 Γεμίζοντας με χρώμα: Εξέταση Σεπτεμβρίου 2024, Θέμα 6 · ★★★ · programming · `exam-2024-sep-q6`
- A25.15 Το Καλό το Μονοπάτι - path: Εξέταση Σεπτεμβρίου 2025, Θέμα 5 · ★★★ · programming · `exam-2025-sep-q5`
- A25.16 Περικύκλωση - encirclement: Εξέταση Ιανουαρίου 2026, Θέμα 5 · ★★★ · programming · `exam-2026-jan-q5`