ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Δείκτες και Πίνακες (Κεφάλαιο 12, ΚΝΚ-2ED)

Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

http://www.cs.ucy.ac.cy/courses/EPL232

Dr. Andreas Aristidou a.aristidou@ieee.org

Περιεχόμενο Διάλεξης 6

- **Δείκτες & Πίνακες**
  - Αριθμητική Δεικτών (Pointer Arithmetic)
    - Χρήση Ονόματος Πίνακα για Δείκτη (Array Name as Pointer)
  - Χρήση Δεικτών για Επεξεργασία Πινάκων (Pointers for Array Processing)
  - Δείκτες και Πολυδιάστατοι Πίνακες (Pointers and Multidimensional Arrays)

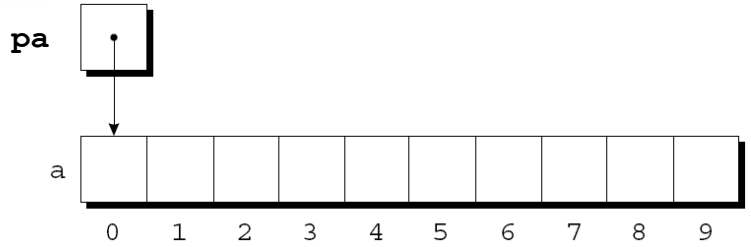
Dr. Andreas Aristidou a.aristidou@ieee.org

Αριθμητική Δεικτών - Pointers Arithmetic

- Στην C υπάρχει ισχυρός δεσμός ανάμεσα στους **δείκτες** και στους **πίνακες**. Οποιαδήποτε πράξη μπορεί να γίνει με **δείκτες πινάκων** (π.χ., a[i]) μπορεί να γίνει και με **δείκτες διευθύνσεων *pa**.
- C μας επιτρέπει να κάνουμε αριθμητικές πράξεις — πρόσθεση και αφαίρεση — σε δείκτες σε στοιχεία πίνακα.
  - Αυτό οδηγεί σε έναν εναλλακτικό τρόπο επεξεργασίας πινάκων (arrays) στις οποίες οι δείκτες διευθύνσεων παίρνουν τη θέση των στοιχείων του πίνακα.
- Η κατανόηση αυτής της σχέσης είναι **κρίσιμη** για την C.

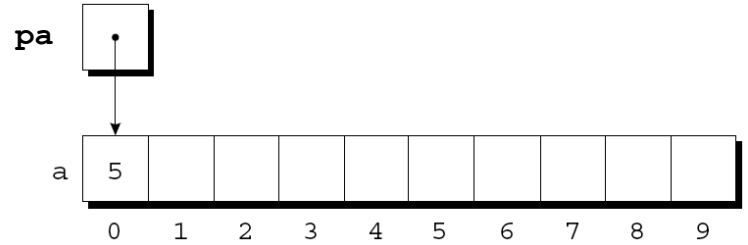
Αριθμητική Δεικτών - Pointers Arithmetic

- Έστω πίνακας `int a[10]`. Η δήλωση αυτή ορίζει ένα μπλοκ 10 διαδοχικών αντικειμένων με ονόματα `a[0]`, `a[1]`, …, `a[9]`.
  - Αν `int *pa = NULL;`
    τότε η ανάθεση `pa = &a[0];`
    κάνει τον **`pa`** να δείχνει το μηδενικό στοιχείο του πίνακα.



- Μπορούμε τώρα να έχουμε πρόσβαση στο `a[0]` μέσω του δείκτη `pa`.
  - Για παράδειγμα, μπορούμε να αποθηκεύσουμε την τιμή 5 στον πίνακα `a[0]` γράφοντας

```
*pa = 5;
```



Αριθμητική Δεικτών - Pointers Arithmetic

- ***Αριθμητική δεικτών***: Αν ο pa είναι δείκτης σε κάποια θέση του πίνακα, τότε οι `pa ± i`, `pa++`, `pa--` είναι επίσης δείκτες
  - **`pa + 1, pa++`** δείκτης στο επόμενο στοιχείο του πίνακα από αυτό που δείχνει ο `pa`.
  - **`pa + i, (pa – i)`** δείκτης στο σημείο του πίνακα που βρίσκεται `i` θέσεις μετά (πριν) από αυτό που δείχνει ο `pa`.

- **Προσοχή:** Η C υποστηρίζει **τρεις** (και μόνο αυτούς τους τρεις) τύπους για αριθμητική δεικτών:
  - Προσθήκη ενός ακέραιου σε έναν δείκτη
  - Αφαίρεση ενός ακέραιου από έναν δείκτη
  - Αφαίρεση ενός δείκτη από έναν άλλο

Dr. Andreas Aristidou a.aristidou@ieee.org

Αριθμητική Δεικτών - Pointers Arithmetic

- Προσθέτοντας ένα ακέραιο `j` στο δείκτη `pa` ουσιαστικά κάνει τον δείκτη να δείχνει σε ένα στοιχείο στον πίνακα που βρίσκεται `j` θέσεις πιο κάτω από εκεί που έδειχνε αρχικά ο `pa`.
- Πιο συγκεκριμένα, αν ο `pa` δείχνει στο στοιχείο του πίνακα `a[i]`, τότε ο `p + j` δείχνει στο στοιχείο `a[i+j]`.
- Ας υποθέσουμε τις ακόλουθες δηλώσεις :

```
int a[10], *p, *q, i;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Παράδειγμα 1

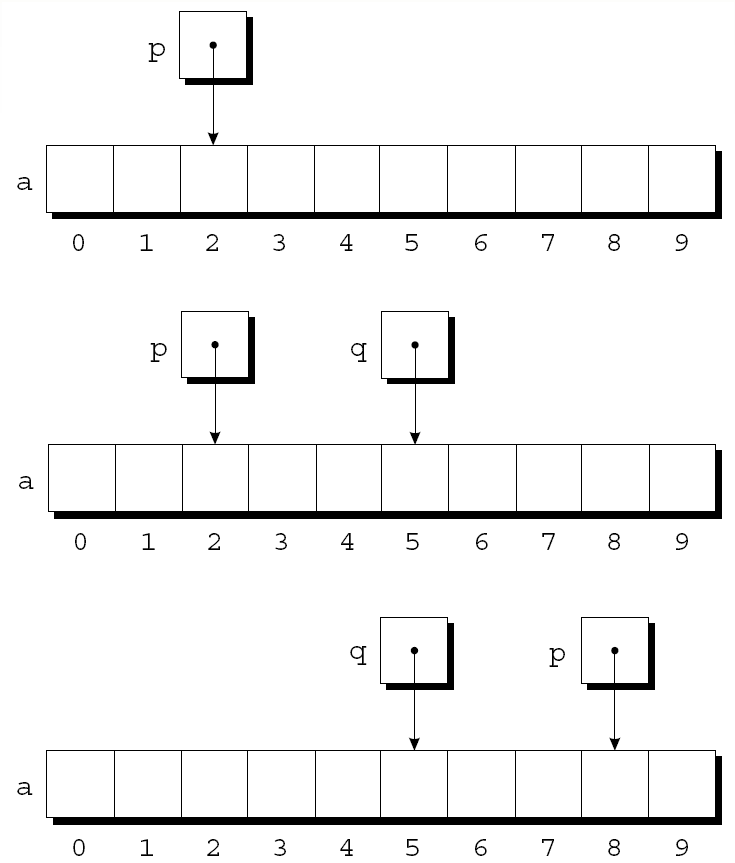
Αριθμητική Δεικτών - Pointers Arithmetic

- Παράδειγμα πρόσθεσης δεικτών:

```
p = &a[2];
```

```
q = p + 3;
```

```
p += 6;
```



Dr. Andreas Aristidou a.aristidou@ieee.org

Παράδειγμα 2

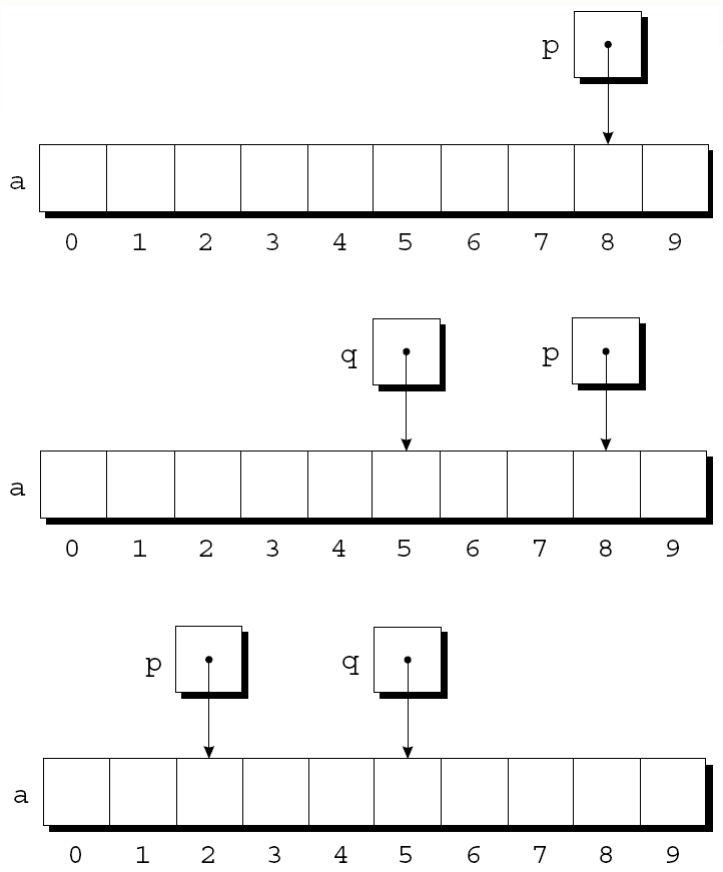
Αριθμητική Δεικτών - Pointers Arithmetic

- Παράδειγμα αφαίρεσης δεικτών:

```
p = &a[8];

q = p - 3;

p -= 6;
```



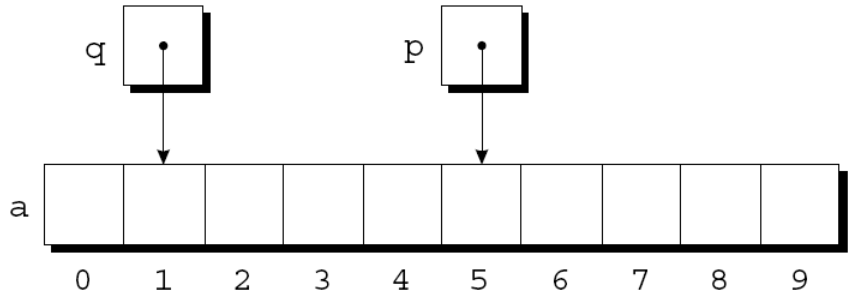
Παράδειγμα 3

Αριθμητική Δεικτών - Pointers Arithmetic

- Όταν ένας δείκτης αφαιρείται από έναν άλλο, το αποτέλεσμα είναι η απόσταση (μετρούμενη σε στοιχεία πίνακα) μεταξύ των δεικτών.
- Εάν ο `p` δείχνει στο `a[i]` και ο `q` δείχνει στο `a[j]`, τότε η αφαίρεση `p – q` είναι ίση με `i – j`.

```
p = &a[5];
q = &a[1];


 i = p - q;    /* i is 4 */
i = q - p;    /* i is -4 */
```



Αριθμητική Δεικτών - Pointers Arithmetic

- Με άλλα λόγια, αν **pa = &a[0],** τότε,

  pa + 2 δείχνει στη τρίτη θέση του πίνακα, δηλ. στο a[2].

  ***(pa + 2)** έχει την τιμή της 3ης θέσης του πίνακα, δηλ. το a[2].

- Εξ' ορισμού, η τιμή μιας μεταβλητής τύπου πίνακα (int a[]) είναι η διεύθυνση του μηδενικού στοιχείου του πίνακα.
  - Επομένως αντί **pa = &a[0]** μπορούμε να γράψουμε **pa = a.**
- Τότε οι **pa** και **a** έχουν τις ίδιες τιμές και μπορούν να χρησιμοποιούνται η μια στη θέση της άλλης:
  - Το **a[i]** είναι ταυτόσημο με τα ***(pa +i)** ή ***(a + i)** ή **pa[i].**
  - Το **&a[i]** είναι ταυτόσημο με τα **a+i, pa+i.**

Dr. Andreas Aristidou a.aristidou@ieee.org

Αριθμητική Δεικτών - Pointers Arithmetic

- Ωστόσο **pa** και **a** **ΔΕΝ** είναι **ταυτόσημα** (sizeof(pa) <> sizeof(a))

  π.χ., `int a[10]; int *pa = NULL;`
  - **sizeof(pa)** = 4 bytes (ILP32) ή 8 bytes (LP64)
  - **sizeof(a)** = μέγεθος πίνακα * μέγεθος τύπου
- Επίσης, δεν μπορώ να μεταβάλω ένα πίνακα (π.χ., a++, a=pa)
  - Δες παράδειγμα στην επόμενη διαφάνεια.

Dr. Andreas Aristidou a.aristidou@ieee.org

Παράδειγμα

Αριθμητική Δεικτών - Pointers Arithmetic

- **Προσοχή:** ο δείκτης είναι μεταβλητή και έτσι **pa = a** και **pa++** είναι **έγκυρες εκφράσεις**. Το **όνομα ενός πίνακα όμως δεν είναι μεταβλητή**, επομένως πράξεις όπως **a = pa** και **a++** δεν είναι έγκυρες!

- Ποιες από τις πιο κάτω εντολές είναι έγκυρες;

```
int t[10] = {0}, i = 5, *p = NULL;
p    = t;
p[2] = 3;              // ίδιο με *(p+2) = 3
++p;
*p   = 14;
*(t+i) = 33;           // ίδιο με t[i] = 33
++t;          => compile error
```

value	0	14	3	0	0	33	0	0	0	0
index	t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]	t[9]

Dr. Andreas Aristidou a.aristidou@ieee.org

Σύγκριση Δεικτών - Comparing Pointers

- Για τη σύγκριση δεικτών μπορούν να χρησιμοποιηθούν όλοι οι γνωστοί **σχεσιακοί τελεστές (<, <=, >, >=)** και οι **τελεστές ισότητας (== and !=).**
  - Ουσιαστικά **συγκρίνονται διευθύνσεις μνήμης** (θυμηθείτε ότι κάθε διεύθυνση **αναφέρεται** σε ένα byte μνήμης)
  - Η χρήση των σχεσιακών τελεστών έχει **νόημα MONO** για **δείκτες** σε **αντικείμενα** του **ίδιου πίνακα**
    - τα οποία βρίσκονται σε **συνεχόμενες διευθύνσεις μνήμης** (άλλες μεταβλητές μπορεί να βρίσκονται σε αυθαίρετες άλλες διευθύνσεις),
- **Παράδειγμα:** το αποτέλεσμα της σύγκρισης εξαρτάται από την απόσταση δυο στοιχείων στον πίνακα., π.χ.,

```
p = &a[5];
q = &a[1];
(p >= q)  => TRUE  (1)
(p <= q)  => FALSE (0)
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Παράδειγμα 4

Αριθμητική Δεικτών & Πίνακες

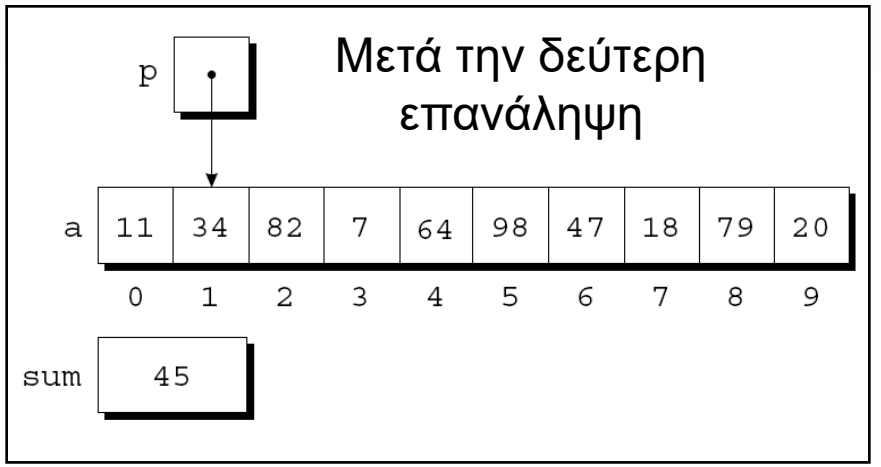
- Τι κάνει το ακόλουθο πρόγραμμα;

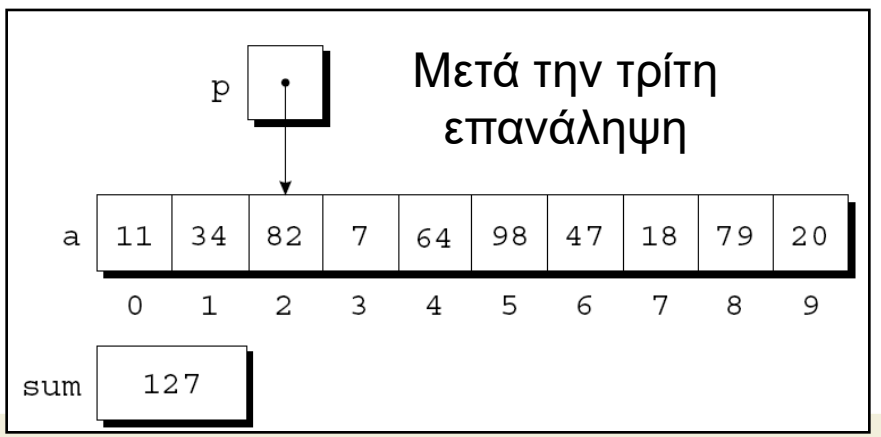
```
#define N 10
…
int a[N], sum, *p = NULL;
…
sum = 0;
for (p = &a[0]; p < &a[N]; p++)
  sum += *p;
```

ή (το αντίστοιχο)

```
for (p = a; p < a + N; p++)
  sum += *p;
```

Το p++ προχωρεί 4 bytes (ILP32-int) σε κάθε κλήση!





Παράδειγμα 5

Αριθμητική Δεικτών & Πίνακες

```
/* Πρόγραμμα αντιστροφής αριθμών με αριθμητική δεικτών */

#include <stdio.h>

#define N 10

int main(void)
{
  int a[N], *p = NULL;

  printf("Enter %d numbers: ", N);
  for (p = a; p < a + N; p++)
    scanf("%d", p); // το p είναι δείκτης άρα δεν θέλει &

  printf("In reverse order:");
  for (p = a + N - 1; p >= a; p--)
    printf(" %d", *p);
  printf("\n");

  return 0;
}
```

For loop με αριθμητική δεικτών

Dr. Andreas Aristidou a.aristidou@ieee.org

Συνδυασμός Τελεστών (* και ++)

- Οι προγραμματιστές C συχνά συνδυάζουν τους τελεστές *** (έμμεσης αναφοράς)** and **++ (αύξησης).**
- Παράδειγμα
  - Μια πρόταση η οποία αναθέτει το `5` στη θέση `a[i]` και στη συνέχεια αυξάνει το `i` κατά `1`:

```
a[i++] = 5;
// a[++i] = 5; Πρώτα αύξηση, μετά ανάθεση
```

  - Η αντίστοιχη έκδοση με δείκτες:

```
*p++ = 5; // Να αποφεύγονται εκφράσεις χωρίς παρενθέσεις
```

  - Το `++` έχει μεγαλύτερη **προτεραιότητα** από το `*` (βλέπε επόμενη διαφάνεια), ωστόσο λόγω της επιθετικότητας του τελεστή ++ ο μεταγλωττιστής το **βλέπει** ως:

```
*(p++) = 5;  // Ισοδύναμο με *p = 5; p++;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Arithmetic Expression Evaluation

Αποτίμηση Αριθμητικών Εκφράσεων

- Πίνακας Τελεστών που συζητήθηκαν μέχρι στιγμής:

Προτεραιότητα	*Όνομα*	*Σύμβολο(α)*	*Εταιρικότητα*
1	increment (postfix)	++	left-to-right
	decrement (postfix)	--	
2	indirect reference	*	**right-to-left**
	address	&	
	increment (prefix)	++	
	decrement (prefix)	--	
	unary plus	+	
	unary minus	-	
3	multiplicative	* / %	left-to-right
4	additive	+ -	left-to-right
5	assignment	= *= /= %= += -=	right-to-left

Full Table: http://www.difranco.net/compsci/C_Operator_Precedence_Table.htm

Dr. Andreas Aristidou a.aristidou@ieee.org

Συνδυασμός Τελεστών (* και ++)

- Πιθανοί συνδυασμοί των * και ++
  - Χρησιμοποιείται **ΠΑΝΤΑ παρενθέσεις** για να αποφύγετε τον πονοκέφαλο των προτεραιοτήτων

A=

	Έκφραση	*Νόημα*
Προχωρεί **Δείκτη**	`*(p++)`	Η τιμή της μεταβλητής A είναι `*p` πριν την μετακίνηση. Στη συνέχεια **προχωρεί** το `p`
	`*(++p)`	**Προχώρησε** το `p` πρώτα. Η τιμή της μεταβλητής A είναι `*p` μετά την μετακίνηση
Αύξηση **Τιμής**	`(*p)++`	Η τιμή της μεταβλητής A είναι `*p` πριν την **αύξηση**. Στη συνέχεια αυξάνεται το `*p`
	`++(*p)`	**Αύξησε** το `*p` πρώτα. Η τιμή της μεταβλητής A είναι `*p` μετά την αύξηση

Εκτός Έκφρασης
p++; Ίδιο με ++p;

Στα Πλαίσια Έκφρασης (Ανάθεσης)
p2 = p++; Διαφορετικό από p2 = ++p;

Συνδυασμός Τελεστών (* και ++)

- Ο πιο κοινός συνδυασμός * και ++ είναι το *p++, που είναι βολικό σε βρόγχους.
- Αντί να γράψουμε

```
for (p = &a[0]; p < &a[N]; p++)
   sum += *p;
```

για να αθροίσουμε τα στοιχεία του πίνακα a, θα μπορούσαμε να γράψουμε

```
p = &a[0];
while (p < &a[N])
   sum += *p++;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Παράδειγμα 6

Πίνακες – Συναρτήσεις - Δείκτες

- Κλήση Συνάρτησης:

```
largest = find_largest(a, N);
```

- Εναλλακτικό Πρότυπο με Δείκτες:

```
int find_largest(int *a, int n)
```

- **Προστασία Στοιχείων** Πίνακα (το a προστατεύεται έτσι και αλλιώς εφόσον έχει τοπική ορατότητα - εμβέλεια)

```
int find_largest(const int a[], int n)
```

ή `int find_largest(const int *a, int n)`

- Η `find_largest` εκτελείται από το a[2] στοιχείο:

```
find_largest(&a[2], N-2);
```

```
int find_largest(int a[], int n) {
  int i, max;
  max = a[0];
  for (i = 1; i < n; i++)
    if (a[i] > max)
      max = a[i];
  return max;
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Χρήση ενός ονόματος πίνακα ως δείκτη

- Ας υποθέσουμε ότι ο πίνακας `a` δηλώνεται ως εξής:

```
int a[10];
```

- Παραδείγματα χρήσης του `a` ως δείκτης:

```
*a = 7;          /* stores 7 in a[0] */
*(a+1) = 12;     /* stores 12 in a[1] */
```

- Γενικά, το `a + i` είναι ίδιο με το `&a[i]`.
  - Και οι δύο αντιπροσωπεύουν ένα δείκτη προς το στοιχείο `i` του `a`.
- Επίσης, `*(a+i)` έχει την ίδια σημασία με το `a[i]`.
  - Αμφότεροι αναπαριστούν το στοιχείο `i`.

Χρήση ενός ονόματος πίνακα ως δείκτη

- Το γεγονός ότι ένα όνομα πίνακα μπορεί να χρησιμεύσει ως δείκτης καθιστά ευκολότερη την γραφή βρόγχων.
- Αρχικός βρόχος:

```
for (p = &a[0]; p < &a[N]; p++)
  sum += *p;
```

- Απλοποιημένη έκδοση:

```
for (p = a; p < a + N; p++)
  sum += *p;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Παράδειγμα

Αντιστροφή μιας σειράς αριθμών

- Το πρόγραμμα `reverse.c` ζητά από τον χρήστη να εισάγει μια σειρά από αριθμούς, και στην συνέχεια την εκτυπώνει σε αντίστροφη σειρά:

```
Enter 10 numbers: 34 82 49 102 7 94 23 11 50 31
In reverse order: 31 50 11 23 94 7 102 49 82 34
```

- Το πρόγραμμα αποθηκεύει τους αριθμούς σε έναν πίνακα καθώς διαβάζονται και, στη συνέχεια, περνά από τον πίνακα ανάποδα, εκτυπώνοντας τα στοιχεία ένα προς ένα.
- Δείτε τις σημειώσεις για εργαστήριο «03_Fundamentals_II_Recitation»

Dr. Andreas Aristidou a.aristidou@ieee.org

Παράδειγμα

Αντιστροφή μιας σειράς αριθμών

reverse.c

```
/* Reverses a series of numbers */

#include <stdio.h>

#define N 10

int main(void)
{
  int a[N], i;

  printf("Enter %d numbers: ", N);
  for (i = 0; i < N; i++)
    scanf("%d", &a[i]);

  printf("In reverse order:");
  for (i = N - 1; i >= 0; i--)
    printf(" %d", a[i]);
  printf("\n");

  return 0;
}
```

reverse3.c

```
/* Reverses a series of numbers (pointer version) */

#include <stdio.h>

#define N 10

int main(void)
{
  int a[N], *p;

  printf("Enter %d numbers: ", N);
  for (p = a; p < a + N; p++)
    scanf("%d", p);

  printf("In reverse order:");
  for (p = a + N - 1; p >= a; p--)
    printf(" %d", *p);
  printf("\n");

  return 0;
}
```

Ορίσματα πίνακα (Ανακεφαλαίωση)

- Όταν μεταβιβάζεται σε μια συνάρτηση, το όνομα ενός πίνακα αντιμετωπίζεται ως δείκτης.
- Παράδειγμα:

```
int find_largest(int a[], int n)
{
  int i, max;

  max = a[0];
  for (i = 1; i < n; i++)
    if (a[i] > max)
      max = a[i];
  return max;
}
```

- Η κλήση της `find_largest`:

```
largest = find_largest(b, N);
```

Ο δείκτης στο πρώτο στοιχείο του `b` ανατίθεται στο `a`. **Ο πίνακας δεν αντιγράφεται.**

Ορίσματα πίνακα (Ανακεφαλαίωση)

- Το γεγονός ότι ένα όρισμα πίνακα αντιμετωπίζεται ως δείκτης έχει κάποιες σημαντικές συνέπειες.
- ***Συνέπεια 1:*** Όταν μια μεταβλητή μεταβιβάζεται σε μια συνάρτηση, η τιμή της αντιγράφεται. Οι αλλαγές στην αντίστοιχη παράμετρο δεν επηρεάζουν τη μεταβλητή.
- Αντίθετα, ένας πίνακας που χρησιμοποιείται ως όρισμα δεν προστατεύεται από τέτοιες αλλαγές.

Ορίσματα πίνακα (Ανακεφαλαίωση)

- Για παράδειγμα, η ακόλουθη συνάρτηση τροποποιεί έναν πίνακα αποθηκεύοντας μηδέν σε καθένα από τα στοιχεία του:

```
void store_zeros(int a[], int n)
{
  int i;

  for (i = 0; i < n; i++)
    a[i] = 0;
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Ορίσματα πίνακα (Ανακεφαλαίωση)

- Για να σιγουρευτείτε ότι δεν θα αλλάξει καμία παράμετρος του πίνακα, μπορείτε να συμπεριλάβετε το `const` κατά τη δήλωση του:

```
int find_largest(const int a[], int n)
{
  …
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Ορίσματα πίνακα (Ανακεφαλαίωση)

- ***Συνέπεια 2:*** Ο απαιτούμενος χρόνος για να περάσει ένας πίνακας σε μια συνάρτηση δεν εξαρτάται από το μέγεθος του πίνακα.
- **Δεν γίνεται αντιγραφή του πίνακα!**

Dr. Andreas Aristidou a.aristidou@ieee.org

Ορίσματα πίνακα (Ανακεφαλαίωση)

- ***Συνέπεια 3:*** Μια παράμετρος του πίνακα μπορεί να δηλωθεί ως δείκτης
- Η `find_largest` θα μπορούσε να οριστεί ως εξής:

```
int find_largest(int *a, int n)
{
  …
}
```

- Δηλώνοντας το `a` σαν ένα δείκτης ισοδυναμεί με τη δήλωση ότι είναι πίνακας. Ο μεταγλωττιστής μεταχειρίζεται τις δύο δηλώσεις σαν να ήταν πανομοιότυπες.

Dr. Andreas Aristidou a.aristidou@ieee.org

Ορίσματα πίνακα (Ανακεφαλαίωση)

- Αν και η δήλωση μιας ***παραμέτρου*** ως πίνακα είναι ίδια με την δήλωση ότι είναι δείκτης, δεν ισχύει το ίδιο και στη περίπτωση μιας ***μεταβλητής***.
- Η ακόλουθη δήλωση αναγκάζει το μεταγλωττιστή να δεσμεύσει χώρο για 10 ακέραιους:

```
int a[10];
```

- Η ακόλουθη δήλωση αναγκάζει το μεταγλωττιστή να δεσμεύσει χώρο για μια μεταβλητή δείκτη:

```
int *a;
```

Ορίσματα πίνακα (Ανακεφαλαίωση)

- Στην τελευταία περίπτωση, ο `a` δεν είναι πίνακας. Η χρησιμοποίηση του σαν πίνακας μπορεί να έχει καταστροφικά αποτελέσματα.
- Για παράδειγμα, η ανάθεση

```
*a = 0;    /*** WRONG ***/
```

  θα αποθηκεύσει το 0 εκεί που δείχνει ο `a`.
- Αφού δεν ξέρουμε που δείχνει ο `a`, το αποτέλεσμα του προγράμματος είναι απροσδιόριστο.

Dr. Andreas Aristidou a.aristidou@ieee.org

Ορίσματα πίνακα (Ανακεφαλαίωση)

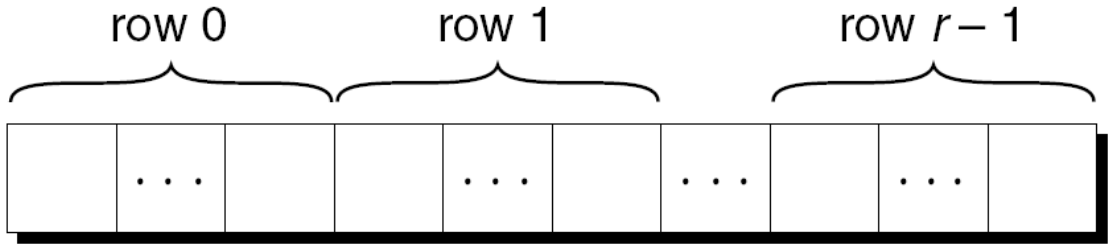
- ***Συνέπεια 4:*** Μια συνάρτηση με μια παράμετρο πίνακα μπορεί να περάσει έναν «κομμάτι» του πίνακα—μια σειρά διαδοχικών στοιχείων.
- Ένα παράδειγμα: Το `find_largest` περνάει το 5º έως το 14º στοιχείο ενός πίνακα `b`:

```
largest = find_largest(&b[5], 10);
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Δείκτες και Πολυδιάστατοι Πίνακες

- Παρόμοιο με τους 1-d πίνακες (arrays), έτσι και στους M-d πίνακες (matrices), οι δείκτες μπορούν να χρησιμοποιηθούν για να δείχνουν σε στοιχεία των πινάκων.
- Εάν ο `p` αρχικά δείχνει στο στοιχείο που βρίσκεται στη γραμμή 0, στήλη 0, μπορούμε να επισκεφτούμε κάθε στοιχείο του πίνακα, αυξάνοντας επανειλημμένα το p.



- Π.χ., μηδενισμός θέσεων πίνακα

```
int a[ROW][COL], *p = NULL, i;
…
for (p = &a[0][0]; p <= &a[ROW-1][COL-1]; p++)
  *p = 0; // διαχείριση στοιχείων με δείκτες
```

Επεξεργασία στοιχείων πολυδιάστατων πινάκων

- Έστω θέλουμε να αρχικοποιήσουμε τα στοιχεία του ακόλουθου πίνακα στο 0:

```
int a[NUM_ROWS][NUM_COLS];
```

- Η προφανής τεχνική θα ήταν η χρήση ένθετων βρόγχων `for` :

```
int row, col;
…
for (row = 0; row < NUM_ROWS; row++)
  for (col = 0; col < NUM_COLS; col++)
    a[row][col] = 0;
```

- Αν δούμε τον `a` ως μια μονοδιάστατη σειρά ακεραίων, ένας βρόγχος είναι αρκετός:

```
int *p;
…
for (p = &a[0][0]; p <= &a[NUM_ROWS-1][NUM_COLS-1]; p++)
  *p = 0;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Επεξεργασία στοιχείων πολυδιάστατων πινάκων

- Μια μεταβλητή δείκτη `p` μπορεί επίσης να χρησιμοποιηθεί για την επεξεργασία των στοιχείων ως μια μόνο σειρά ενός δισδιάστατου πίνακα.
- Για να επισκεφθείτε τα στοιχεία της γραμμής `i`, θα ορίσουμε το `p` να δείχνει το στοιχείο 0 στην γραμμή `i` του πίνακα `a`:

```
p = &a[i][0];
```

ή θα μπορούσαμε απλά να γράψουμε

```
p = a[i];
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Επεξεργασία στοιχείων πολυδιάστατων πινάκων

- Για κάθε δισδιάστατο πίνακα `a`, η έκφραση `a[i]` είναι ένας δείκτης στο πρώτο στοιχείο της γραμμής `i`.
- Για να δείτε γιατί αυτό λειτουργεί, θυμηθείτε ότι `a[i]` ισοδυναμεί με `*(a + i)`.
- Επομένως, `&a[i][0]` είναι το ίδιο με `&(*(a[i] + 0))`, που ισοδυναμεί με `&*a[i]`.
- Αυτό είναι το ίδιο με το `a[i]`, αφού οι τελεστές `&` και `*` αλληλοαναιρούνται.

Dr. Andreas Aristidou a.aristidou@ieee.org

Επεξεργασία στοιχείων πολυδιάστατων πινάκων

- Ένας βρόχος που καθαρίζει τη γραμμή `i` του πίνακα `a`:

```
int a[NUM_ROWS][NUM_COLS], *p, i;
…
for (p = a[i]; p < a[i] + NUM_COLS; p++)
  *p = 0;
```

- Αφού ο `a[i]` είναι δείκτης που δείχνει στην γραμμή `i` του πίνακα `a`, μπορούμε να περάσουμε την `a[i]` σε μια συνάρτηση που αναμένει ως όρισμα έναν πίνακα μίας διάστασης.
- Με άλλα λόγια, μια συνάρτηση που έχει σχεδιαστεί για να λειτουργεί με μονοδιάστατους πίνακες θα λειτουργεί επίσης και με μια γραμμή που ανήκει σε έναν πίνακα δύο διαστάσεων.

Dr. Andreas Aristidou a.aristidou@ieee.org

Επεξεργασία στοιχείων πολυδιάστατων πινάκων

- Έστω η `find_largest`, που σχεδιάστηκε αρχικά για να βρει το μεγαλύτερο στοιχείο ενός μονοδιάστατου πίνακα.
- Μπορούμε να χρησιμοποιήσουμε τη `find_largest` για να προσδιορίσουμε το μεγαλύτερο στοιχείο στη γραμμή `i` ενός δισδιάστατου πίνακα `a`:

```
largest = find_largest(a[i], NUM_COLS);
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Επεξεργασία στοιχείων πολυδιάστατων πινάκων

- Η επεξεργασία των στοιχείων μιας *στήλης* σε ένα πίνακα δύο διαστάσεων δεν είναι τόσο εύκολη, επειδή οι πίνακες αποθηκεύονται ανά *γραμμή* και όχι ανά *στήλη*.
- Ένας βρόγχος που καθαρίζει τη στήλη `i` του πίνακα `a`:

```
int a[NUM_ROWS][NUM_COLS], (*p)[NUM_COLS], i;
…
for (p = &a[0]; p < &a[NUM_ROWS]; p++)
  (*p)[i] = 0;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Επεξεργασία στοιχείων πολυδιάστατων πινάκων

- Το όνομα *οποιουδήποτε πίνακα* μπορεί να χρησιμοποιηθεί ως δείκτης, ανεξάρτητα από το πόσες διαστάσεις έχει.
- Παράδειγμα:

```
int a[NUM_ROWS][NUM_COLS];
```

  Ο `a` δεν δείχνει στο `a[0][0]`, αλλά στο `a[0]`.
- Η C θεωρεί το `a` σαν έναν μονοδιάστατο πίνακα.
- Όταν χρησιμοποιηθεί σαν δείκτης, ο `a` είναι `int (*)[NUM_COLS]` (δείκτης σε έναν ακέραιο πίνακα μήκους `NUM_COLS`).

Dr. Andreas Aristidou a.aristidou@ieee.org

Χρήση του ονόματος ενός πολυδιάστατου πίνακα ως δείκτη

- Γνωρίζοντας πως ο `a` δείχνει στο `a[0]` είναι χρήσιμο για την απλούστευση των βρόγχων που επεξεργάζονται τα στοιχεία ενός δισδιάστατου πίνακα.
- Οπότε, αντί να γράψετε

```
for (p = &a[0]; p < &a[NUM_ROWS]; p++)
  (*p)[i] = 0;
```

για να καθαρίσετε την στήλη `i` του πίνακα `a`, μπορείτε να γράψετε

```
for (p = a; p < a + NUM_ROWS; p++)
  (*p)[i] = 0;
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Χρήση του ονόματος ενός πολυδιάστατου πίνακα ως δείκτη

- Μπορούμε να "ξεγελάσουμε" μια συνάρτηση ώστε να εκλαμβάνει ένα πολυδιάστατο πίνακα ως μονοδιάστατο.
- Μια πρώτη προσπάθεια χρήσης της `find_largest` να βρει το μεγαλύτερο στοιχείο της `a`:

```
largest = find_largest(a, NUM_ROWS * NUM_COLS);
  /* WRONG */
```

  Αυτό είναι λάθος, επειδή ο τύπος `a` είναι `int (*)[NUM_COLS]` αλλά η `find_largest` αναμένει ένα όρισμα τύπου `int *`.

- Η σωστή κλήση είναι:

```
largest = find_largest(a[0], NUM_ROWS * NUM_COLS);
```

  `a[0]` παραπέμπει στο στοιχείο 0 στη σειρά 0 και έχει τύπο `int *` (μετά τη μετατροπή από το πρόγραμμα μεταγλώττισης).

Παράδειγμα

m-D Πίνακες + Συναρτήσεις + Δείκτες

Ας δούμε **4 διαφορετικές παραλλαγές** υλοποίησης της sum() των στοιχείων ενός 2D πίνακα

```
#define ROW 5
#define COL 5
int sum_two_dimensional_array(const int [][COL]);

int sum_two_dimensional_array(const int a[][COL]) {
    int i, j, sum = 0;

    for(i=0; i<ROW; i++)
        for (j=0; j<COL; j++)
            sum+=a[i][j];
    return sum;
}

int main() {
    const int a[][COL] = {
        {1,1,1,1,1}, {2,2,2,2,2}, {3,3,3,3,3}, {4,4,4,4,4}, {5,5,5,5,5}
    }; // sum = 75

    printf("sum:%d\n", sum_two_dimensional_array(a));
   return 0;
}
```

Λύση 1: Με defined ROW + COL (λογική που έχουμε χρησιμοποιήσει εκτενώς στο παρελθόν)

Dr. Andreas Aristidou a.aristidou@ieee.org

Παράδειγμα

m-D Πίνακες + Συναρτήσεις + Δείκτες

Λύση 2: Με COL, Χωρίς define ROW

```
#define COL 5

int sum_two_dimensional_array(const int [][COL], int);

int sum_two_dimensional_array(const int a[][COL], int ROW) {
    int i, j, sum = 0;

    for(i=0; i<ROW; i++)
        for (j=0; j<COL; j++)
            sum+=a[i][j];
    return sum;
}
int main()
{   int sum = 0;
    const int a[][COL] = {
        {1,1,1,1,1}, {2,2,2,2,2}, {3,3,3,3,3}, {4,4,4,4,4}, {5,5,5,5,5}
    }; // sum = 75 (5x15)

  sum = sum_two_dimensional_array(a, sizeof(a)/sizeof(a[0]));
  return 0;
}
```

Αριθμός Γραμμών

Μέγεθος πίνακα

Μέγεθος (bytes) γραμμής πίνακα

Διαχείριση Στηλών

Δείκτες και Πολυδιάστατοι Πίνακες

- Η επεξεργασία **στηλών** είναι κάπως πιο **περίπλοκη** εφόσον οι πίνακες αποθηκεύονται **ανά γραμμή** (όχι ανά στήλη).
- **Παράδειγμα:** Ένα loop που αναθέτει την τιμή 0 στην στήλη `i` του πίνακα `a`:

```
  int a[ROW][COL], (*p)[COL] = NULL, i=2;
  …                          // Διεύθυνση γραμμής 0
  for (p = &a[0]; p < &a[ROW]; p++)
    (*p)[i] = 0; //
ή
  for (p = a; p < a + ROWS; p++)
    (*p)[i] = 0;
```

Δείκτης σε ένα ROW (πλάτους COL)

το p++ προχωρεί ROW θέσεις!!!!

- Το `(*p)[i]` χρειάζεται παρενθέσεις εφόσον η προτεραιότητα του `[]` είναι ψηλότερη από αυτή του `*`.
  - Εναλλακτικά θα ήταν **πίνακας δεικτών** που θα δούμε αργότερα
- Ακολουθεί παράδειγμα χρήσης του (*p)[i] …

Dr. Andreas Aristidou a.aristidou@ieee.org

Διαχείριση Στηλών

Δείκτες και Πολυδιάστατοι Πίνακες

```
int main() // Πρόγραμμα Μηδενισμού Στήλης με Δείκτης
{    int sum = 0;
     int a[][COL] = {
          {1,1,1,1,1}, {2,2,2,2,2},
           {3,3,3,3,3}, {4,4,4,4,4}, {5,5,5,5,5}
     }; // sum = 75 (5x15)

     int (*p)[COL] = NULL; // δείκτης σε ROW (πλάτους COL)

     int ROW = sizeof(a)/sizeof(a[0]);

     int i = 2; // column to change
     for (p = &a[0]; p < &a[ROW]; p++)
          (*p)[i] = 0;

     print_two_dimensional_array(a, ROW);
     return 0;
}
```

Αριθμός γραμμών

Μέγεθος (bytes) πίνακα / Μέγεθος (bytes) γραμμής πίνακα

το p++ προχωρεί COL θέσεις

INPUT	OUTPUT
1,1,1,1,1, 2,2,2,2,2, 3,3,3,3,3, 4,4,4,4,4, 5,5,5,5,5,	1,1,0,1,1, 2,2,0,2,2, 3,3,0,3,3, 4,4,0,4,4, 5,5,0,5,5,

Dr. Andreas Aristidou a.aristidou@ieee.org

Παράδειγμα

m-D Πίνακες + Συναρτήσεις + Δείκτες

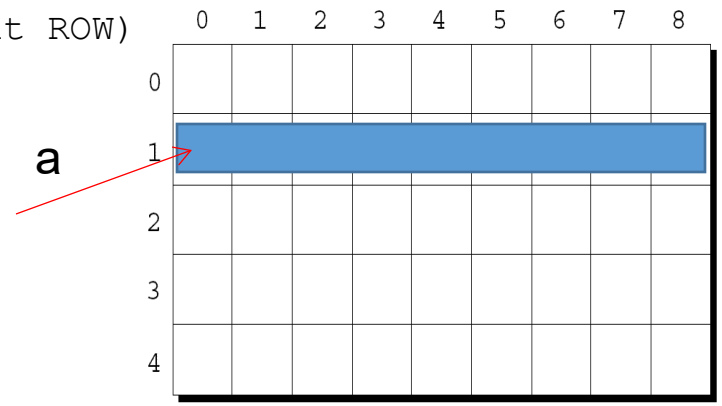
Ενδιάμεση Λύση: Ίδιο με Προηγούμενο αλλά με Δείκτη σε Γραμμή στο πρότυπο

```
#define COL 5


void sum_two_dimensional_array(const int (*)[COL], int);


int sum_two_dimensional_array(const int (*a)[COL], int ROW)
    int i, j, sum = 0;

    for(i=0; i<ROW; i++)
        for (j=0; j<COL; j++)
            sum+=a[i][j];
    return sum;
}
int main()
{   ...
  sum = sum_two_dimensional_array(a, sizeof(a)/sizeof(a[0]));
  ...
}
```



Ίδια κλήση με προηγούμενο παράδειγμα

Παράδειγμα

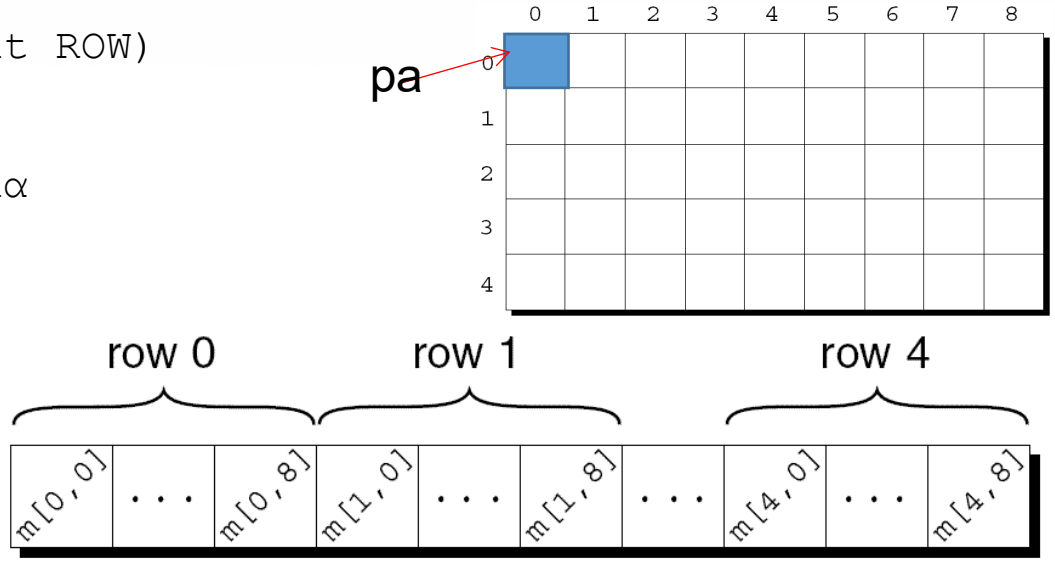
m-D Πίνακες + Συναρτήσεις + Δείκτες

Λύση 3: Δείκτη σε Γραμμή + Απαριθμητό βρόχο

```
#define COL 5
...
int sum_two_dimensional_array(int (*a)[COL], int ROW)
{
     int i, sum = 0;
     int *pa = &a[0][0]; // πρώτο στοιχείο πίνακα

     for(i=0; i < ROW*COL; i++) {
        sum += *pa;
       pa++;
     }
     return sum;
}

int main()
{   ...
   sum = sum_two_dimensional_array(a, sizeof(a)/sizeof(a[0]));
   ...
}
```



Παράδειγμα

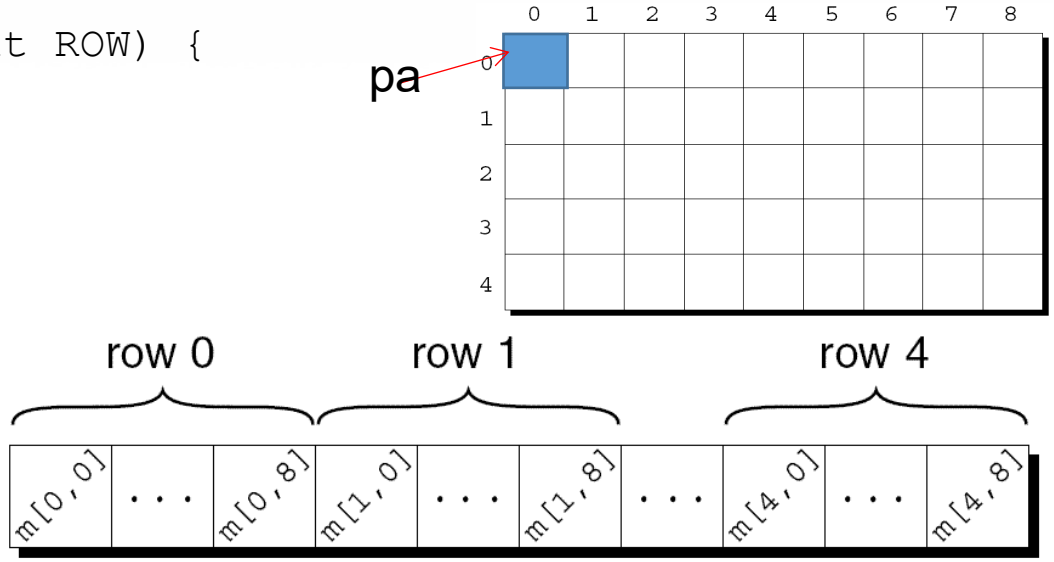
m-D Πίνακες + Συναρτήσεις + Δείκτες

Λύση 4: ΙΔΙΟ με πριν αλλά με Αριθμητική Δεικτών στο for

```
#define COL 5
...
int sum_two_dimensional_array(int (*a)[COL], int ROW) {
    int sum = 0;
    int *pa = NULL;

    for(pa=a[0]; pa < a[0] + ROW*COL; pa++)
        sum += *pa;

    return sum;
}
```



Σημείωση:

Το ακόλουθο είναι λάθος, εφόσον το p είναι **int *** ενώ το **a** είναι **[][] (2D)**

~~`for(p=a; p < a + ROW*COL; p++)`~~

Επισκόπηση: Πίνακες Δεικτών

- Σε όλα τα προηγούμενα παραδείγματα το **COL** ήταν προσδιορισμένο ως **σταθερά**.
- Τι γίνεται εάν **ΔΕΝ γνωρίζουμε** το COL εκ’ των πρότερων;
- Τότε μπορούμε να χρησιμοποιήσουμε **Πίνακες Δεικτών** (που υλοποιούνται με **Δείκτες σε Δείκτες **p**) και τους οποίους θα δούμε στην ερχόμενη διάλεξη.
  - Δεξιά: Παράδειγμα Πίνακα Δεικτών σε Μήνες (κάθε COL έχει διαφορετικό πλάτος)