



ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Διάλεξη 6: Δείκτες και Πίνακες

(Κεφάλαιο 12, ΚΝΚ-2ΕΔ)

Δημήτρης Ζεϊναλιπούρ

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

Περιεχόμενο Διάλεξης 6

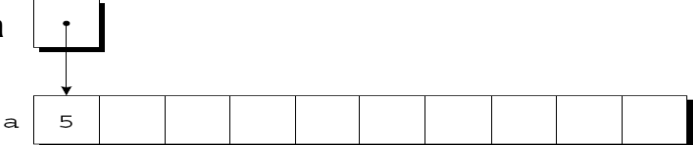


- **Δείκτες & Πίνακες**

- 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)

Αριθμητική Δεικτών (Pointers Arithmetic)



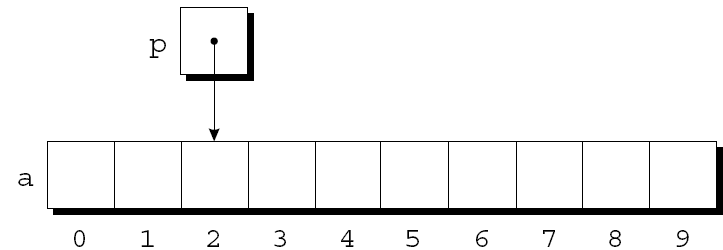
- Στην C υπάρχει ισχυρός δεσμός ανάμεσα στους **δείκτες** και στους **πίνακες**. Οποιαδήποτε πράξη μπορεί να γίνει με **δείκτες πινάκων** (π.χ., $a[i]$) μπορεί να γίνει και με **δείκτες διευθύνσεων** $*pa$.
- Έστω πίνακας $int\ a[10]$. Η δήλωση αυτή ορίζει ένα μπλοκ 10 διαδοχικών αντικειμένων με ονόματα $a[0]$, $a[1]$, ..., $a[9]$.
 - Αν `int *pa = NULL;` 
τότε η ανάθεση `pa = &a[0];`
κάνει τον pa να δείχνει το μηδενικό στοιχείο του πίνακα.
- **Αριθμητική δεικτών**: Αν ο pa είναι δείκτης σε κάποια θέση του πίνακα, τότε οι $pa \pm i$, $pa++$, $pa--$ 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 .

Αριθμητική Δεικτών (Παράδειγμα 1)

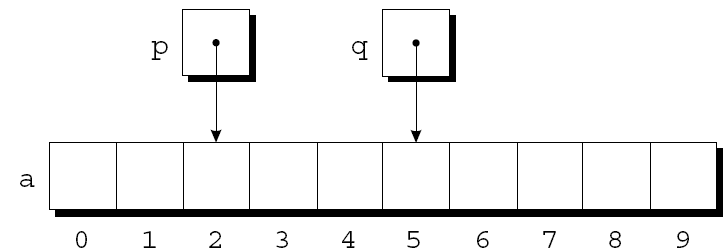


- Παράδειγμα πρόσθεσης δεικτών:

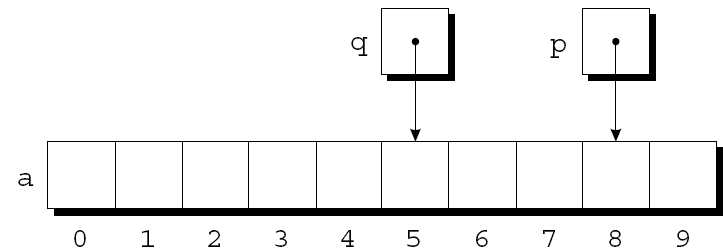
```
p = &a[2];
```



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



```
p += 6;
```



Αριθμητική Δεικτών (Pointers Arithmetic)



- Έτσι, αν $pa = \&a[0]$, τότε για παράδειγμα,
 - $pa + 2$ δείχνει στη τρίτη θέση του πίνακα, δηλ. στο $a[2]$.
 - $*(pa + 2)$ έχει την τιμή της 3ης θέσης του πίνακα, δηλ. το $a[2]$.
- Έξ' 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 $a+i$, $pa+i$.
- Ωστόσο pa και a ΔΕΝ είναι ταυτόσημα ($sizeof(pa) \neq sizeof(a)$)
π.χ., `int a[10]; int *pa = NULL;`
 - $sizeof(pa) = 4$ bytes (ILP32) ή 8 bytes (LP64)
 - $sizeof(a) = \text{μέγεθος πίνακα} * \text{μέγεθος τύπου}$
- Επίσης, δεν μπορώ να μεταβάλω ένα πίνακα (π.χ., $a++$, $a=pa$)
 - Δες παράδειγμα στην επόμενη διαφάνεια.

Αριθμητική Δεικτών (Παράδειγμα 2)



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

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 ΜΟΝΟ** 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.,

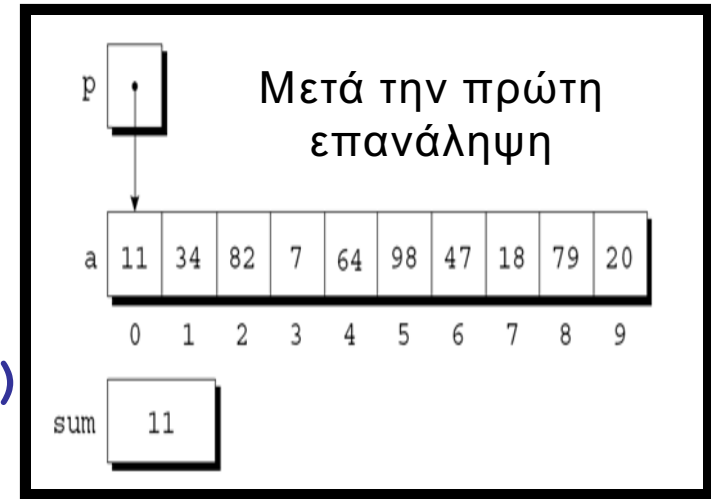
```
p = &a[5];  
q = &a[1];  
(p >= q) => TRUE (1)  
(p <= q) => FALSE (0)
```

Αριθμητική Δεικτών & Πίνακες (Παράδειγμα 3)



- 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;
```

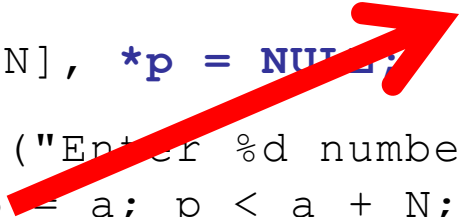
Το `p++` προχωρεί 4 bytes (ILP32-int) σε κάθε κλήση!

Παράδειγμα (1-d Πίνακες + Αριθμητική Δεικτών)



```
/* 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



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++;
```

Αποτίμηση Αριθμητικών Εκφράσεων (Arithmetic Expression Evaluation)



- Πίνακας Τελεστών που συζητήθηκαν μέχρι στιγμής:

<i>Προτεραιότητα</i>	<i>Όνομα</i>	<i>Σύμβολο(α)</i>	<i>Εταιρικότητα</i>
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

Συνδυασμός Τελεστών (* και ++)



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

	Έκφραση	Νόημα
Προχωρεί Δείκτη	* (p++) →	η τιμή της μεταβλητής A είναι *p πριν την μετακίνηση. στη συνέχεια <u>προχωρεί</u> το p
	* (++p) →	<u>προχώρησε</u> 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 <u>αύξηση</u> . στη συνέχεια αυξάνεται το *p
	++ (*p)	<u>αύξησε</u> 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;

Πίνακες – Συναρτήσεις - Δείκτες

- 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;
}
```

- 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, δείτε διαφάνεια 5.30)

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

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

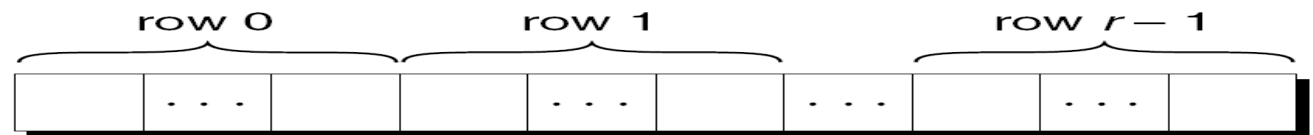
- Η `find_largest` εκτελείται από το `a[2]` στοιχείο:

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

Δείκτες και Πολυδιάστατοι Πίνακες (Διαχείριση Γραμμών)



- 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.



- 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
```

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 (5x15)
```

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

```
    return 0;
```

```
}
```

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

Παράδειγμα

(m-D Πίνακες + Συναρτήσεις + Δείκτες)



```
#define COL 5
```

Λύση 2: Με COL, Χωρίς define ROW

```
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

Δείκτες και Πολυδιάστατοι Πίνακες (Διαχείριση Στηλών)



- Επεξεργασία **στηλών** είναι κάπως πιο **περίπλοκη** εφόσον οι πίνακες αποθηκ. **ανά γραμμή** (όχι ανά στήλη).
- **Παράδειγμα:** Ένα loop που αναθέτει την τιμή 0 στην στήλη *i* του πίνακα *a*: *Δείκτης σε ένα ROW (πλάτους COL)*

```
int a[ROW][COL], (*p)[COL] = NULL, i=2;
...
for (p = &a[0]; p < &a[ROW]; p++)
    (*p)[i] = 0; // το p++ προχωρεί COL θέσεις!!!!
ή
for (p = a; p < a + ROWS; p++)
    (*p)[i] = 0;
```

- 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] ...`

Δείκτες και Πολυδιάστατοι Πίνακες (Διαχείριση Στηλών)



```
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)
```

INPUT

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

OUTPUT

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

Αριθμός γραμμών

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

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

Μέγεθος (bytes) πίνακα

Μέγεθος (bytes) γραμμής πίνακα

```
int i = 2; // column to change
```

```
for (p = &a[0]; p < &a[ROW]; p++)
```

```
    (*p)[i] = 0;
```

το p++ προχωρεί COL θέσεις

```
print_two_dimensional_array(a, ROW);
```

```
return 0;
```

Παράδειγμα

(m-D Πίνακες + Συναρτήσεις + Δείκτες)



```
#define COL 5
```

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

```
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;
```

```
}
```

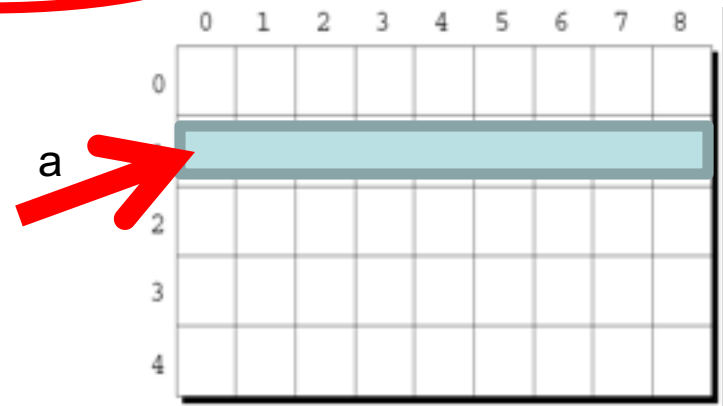
```
int main()
```

```
{    ...
```

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

```
    ...
```

```
}
```



Ίδια κλήση με προηγούμενο
παράδειγμα

Παράδειγμα

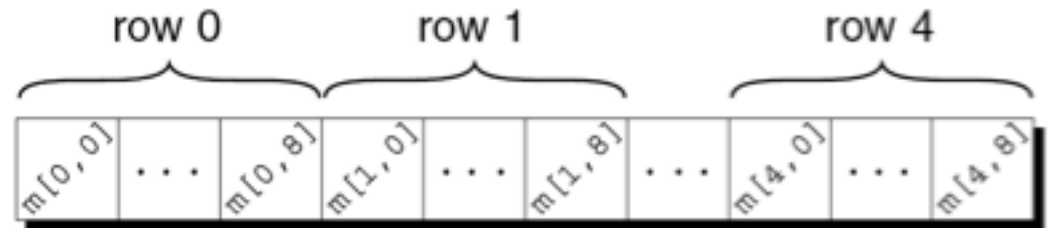
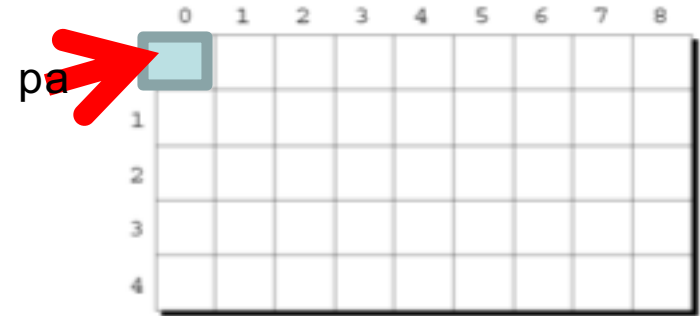


(m-D Πίνακες + Συναρτήσεις + Δείκτες)

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]));
    ...
}
```

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 7.
 - Δεξιά: Παράδειγμα Πίνακα Δεικτών σε Μήνες (κάθε COL έχει διαφορετικό πλάτος)

