

# ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΥΠΟΛΟΓΙΣΤΩΝ & ΥΠΟΛΟΓΙΣΤΙΚΗ ΦΥΣΙΚΗ

Μέρος 6ο

ΝΙΚΟΛΑΟΣ ΣΤΕΡΓΙΟΥΛΑΣ



ΤΜΗΜΑ ΦΥΣΙΚΗΣ

ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

# ΠΙΝΑΚΕΣ

---

Ένας **πίνακας** στη C είναι μία **δομή δεδομένων** που αποτελείται από στοιχεία του ίδιου τύπου (π.χ. πίνακας ακεραίων αριθμών, πίνακας πραγματικών αριθμών, πίνακας χαρακτήρων, ...).

Όλοι οι πίνακες **δεσμεύουν συνεχόμενες θέσεις στη μνήμη** (στην περιοχή μνήμης που ονομάζεται **στοίβα** ή **stack**) του υπολογιστή και διακρίνονται σε πίνακες μίας διάστασης και πίνακες πολλών διαστάσεων.

Συνηθέστερα είδη είναι οι **μονοδιάστατοι** και οι **διδιάστατοι** πίνακες.

# ΟΡΙΣΜΟΣ ΜΟΝΟΔΙΑΣΤΑΤΟΥ ΠΙΝΑΚΑ

---

Για να ορίσουμε έναν **μονοδιάστατο πίνακα** πρέπει να δηλώσουμε το όνομα του πίνακα, τον τύπο δεδομένων των στοιχείων του πίνακα και το πλήθος των στοιχείων του πίνακα.

Ο γενικός ορισμός είναι:

**τύπος\_δεδομένων** **όνομα\_πίνακα** [**πλήθος\_στοιχείων**]

Παραδείγματα:

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

```
double b[100];
```

```
char c[30];
```

# ΔΕΙΚΤΗΣ ΘΕΣΗΣ

---

Όταν ανακαλούμε ένα συγκεκριμένο στοιχείο του πίνακα, π.χ. ως `a[3]` τότε ο αριθμός μέσα στις αγκύλες είναι ο **δείκτης θέσης** του στοιχείου στον πίνακα.

ΠΡΟΣΟΧΗ: αν ορίσουμε `int a[10]` τότε τα στοιχεία του πίνακα είναι τα: `a[0], a[1], ..., a[9]` (!)

Δηλαδή, η **βάση αρίθμησης των στοιχείων είναι 0** (αυτό συμβαίνει στις περισσότερες γλώσσες - μια εξαίρεση αποτελεί η Fortran που έχει βάση το αρίθμησης το 1).

Αν θέλουμε να έχουμε στοιχεία `a[1]` έως `a[10]`, τότε ορίζουμε `int a[11]` και αγνοούμε το `a[0]`.

# ΠΑΡΑΔΕΙΓΜΑ

---

Μια καλή πρακτική είναι η εξής:

Έστω ότι θέλουμε να δηλώσουμε έναν πίνακα που να έχει ακέραια στοιχεία π.χ. `a[1]` έως `a[10]`.

Δηλώνουμε το μέγεθος του πίνακα ως σταθερά στην αρχή του προγράμματος:

```
#define SIZE 10
```

Ορίζουμε τον πίνακα ως:

```
int a[SIZE+1];
```

και απλά αγνοούμε το στοιχείο `a[0]`.

# ΠΑΡΑΔΕΙΓΜΑ

```
#include <stdio.h>

#define SIZE 5

int main(void)
{
    int i;

    double A[SIZE+1];

    printf("A[%d] = %f\n", 0, A[0]);
    printf("\n");

    for(i=1; i<=SIZE; i++)
    {
        A[i] = i*i;
        printf("A[%d] = %f\n", i, A[i]);
    }

    printf("\nA[%d] = %f\n", SIZE+1, A[SIZE+1]);

    return 0;
}
```

# ΠΑΡΑΔΕΙΓΜΑ

$A[0] = 0.000000$  ← ( αγνοούμε το στοιχείο  $A[0]$  )

$A[1] = 1.000000$

$A[2] = 4.000000$

$A[3] = 9.000000$

$A[4] = 16.000000$

$A[5] = 25.000000$

← ( χρήσιμα στοιχεία  
 $A[1]$  έως  $A[SIZE]$  )

$A[6] = 0.000000$  ← ( υπέρβαση ορίων δείκτη θέσης! )

# ΠΑΡΑΔΕΙΓΜΑ

```
#include <stdio.h>

#define SIZE 5

int main(void)
{
    int i;

    double number, A[SIZE+1];

    for(i=1; i<=SIZE; i++)
    {
        printf("Δώσε τον %do αριθμό: ", i);
        scanf("%lf", &number);
        A[i] = number;
    }

    printf("\n");

    for(i=SIZE; i>=1; i--)
        printf("O %dos αριθμός είναι: %f\n", i, A[i]);
}
```



# ΠΑΡΑΔΕΙΓΜΑ

---

Δώσε τον 1ο αριθμό: 3.2  
Δώσε τον 2ο αριθμό: 4.3  
Δώσε τον 3ο αριθμό: 9.81  
Δώσε τον 4ο αριθμό: 34.21  
Δώσε τον 5ο αριθμό: 43.84

0 5ος αριθμός είναι: 43.840000  
0 4ος αριθμός είναι: 34.210000  
0 3ος αριθμός είναι: 9.810000  
0 2ος αριθμός είναι: 4.300000  
0 1ος αριθμός είναι: 3.200000

# ΟΡΙΣΜΟΣ ΔΙΔΙΑΣΤΑΤΟΥ ΠΙΝΑΚΑ

---

Για να ορίσουμε έναν **διδιάστατο πίνακα** πρέπει να δηλώσουμε το όνομα του πίνακα, τον τύπο δεδομένων των στοιχείων του πίνακα, το πλήθος των γραμμών και το πλήθος των στηλών του πίνακα.

Ο γενικός ορισμός είναι:

**τύπος\_δεδομένων** όνομα\_πίνακα [πλήθος\_γραμμών] [πλήθος\_στηλών]

Παραδείγματα:

```
int a[10][20];
```

```
double b[100][200];
```

```
char c[30][40];
```

# ΠΑΡΑΔΕΙΓΜΑ ΟΡΙΣΜΟΥ ΔΙΔΙΑΣΤΑΤΟΥ ΠΙΝΑΚΑ

Αν έχουμε ορίσει τον διδιάστατο πίνακα  $a[3][4]$ , τότε τα στοιχεία του είναι:

	Στήλη 0	Στήλη 1	Στήλη 2	Στήλη 3
Γραμμή 0	$a[0][0]$	$a[0][1]$	$a[0][2]$	$a[0][3]$
Γραμμή 1	$a[1][0]$	$a[1][1]$	$a[1][2]$	$a[1][3]$
Γραμμή 2	$a[2][0]$	$a[2][1]$	$a[2][2]$	$a[2][3]$

Όνομα πίνακα

Δείκτης γραμμής

Δείκτης στήλης

Παρατηρούμε πως και οι διδιάστατοι πίνακες έχουν αρίθμηση δεικτών με βάση το 0.

# ΑΠΟΔΟΣΗ ΑΡΧΙΚΩΝ ΤΙΜΩΝ

---

Στον αρχικό ορισμό του πίνακα `a[3][4]` μπορούμε να συμπεριλάβουμε την εκχώρηση αρχικών τιμών:

```
int a[3][4] = { {0, 1, 2, 3},  
               {4, 5, 6, 7},  
               {8, 9, 10, 11} }
```

έχοντας πάντα υπόψη ότι στον παραπάνω ορισμό το πρώτο στοιχείο είναι το `a[0][0]` και το τελευταίο είναι το `a[2][3]` !

# ΠΑΡΑΔΕΙΓΜΑ ΜΕ ΒΑΣΗ ΤΟ 1

---

Έστω ότι θέλουμε να ορίσουμε έναν διδιάστατο πίνακα με ακέραιες τιμές και *χρήσιμα* στοιχεία `a[1][1]` έως `a[2][3]`.

Δηλώνουμε στην αρχή του προγράμματος:

```
#define XSIZE 2  
#define YSIZE 3
```

ορίζουμε τον πίνακα ως:

```
int a[XSIZE+1][YSIZE+1];
```

και απλά αγνοούμε τα στοιχεία της γραμμής `a[0][ ]`  
καθώς και τα στοιχεία της στήλης `a[ ][0]`.

# ΠΑΡΑΔΕΙΓΜΑ

Δηλαδή:

( αγνοούμε τη γραμμή  $a[0][ ]$  )

	Στήλη 0	Στήλη 1	Στήλη 2	Στήλη 3
Γραμμή 0	$a[0][0]$	$a[0][1]$	$a[0][2]$	$a[0][3]$
Γραμμή 1	$a[1][0]$	$a[1][1]$	$a[1][2]$	$a[1][3]$
Γραμμή 2	$a[2][0]$	$a[2][1]$	$a[2][2]$	$a[2][3]$

Όνομα πίνακα

Δείκτης γραμμής

Δείκτης στήλης

( αγνοούμε τη στήλη  $a[ ][0]$  )

( χρήσιμα στοιχεία  $a[1][1]$  έως  $a[2][3]$  )

# ΕΚΧΩΡΙΣΗ ΤΙΜΩΝ ΣΕ ΠΙΝΑΚΑ

---

Έστω ότι ορίσαμε έναν διδιάστατο πίνακα με τις εντολές

```
#define XSIZE 2  
#define YSIZE 3
```

```
int a[XSIZE+1][YSIZE+1];
```

Μπορούμε να εκχωρίσουμε τιμές με διπλό βρόχο. Π.χ.

```
int i, j;
```

```
for ( i = 1 ; i <= XSIZE ; i++ )  
    for ( j = 1 ; j <= YSIZE ; j++ )  
        a[i,j] = i*i + j*j;
```

# ΠΑΡΑΔΕΙΓΜΑ

```
#include <stdio.h>

#define XSIZE 2
#define YSIZE 3

int main(void)
{
    int i, j, a[XSIZE+1][YSIZE+1];

    for(i=1; i<=XSIZE; i++)
        for(j=1; j<=YSIZE; j++)
            a[i][j] = i*i + j*j;

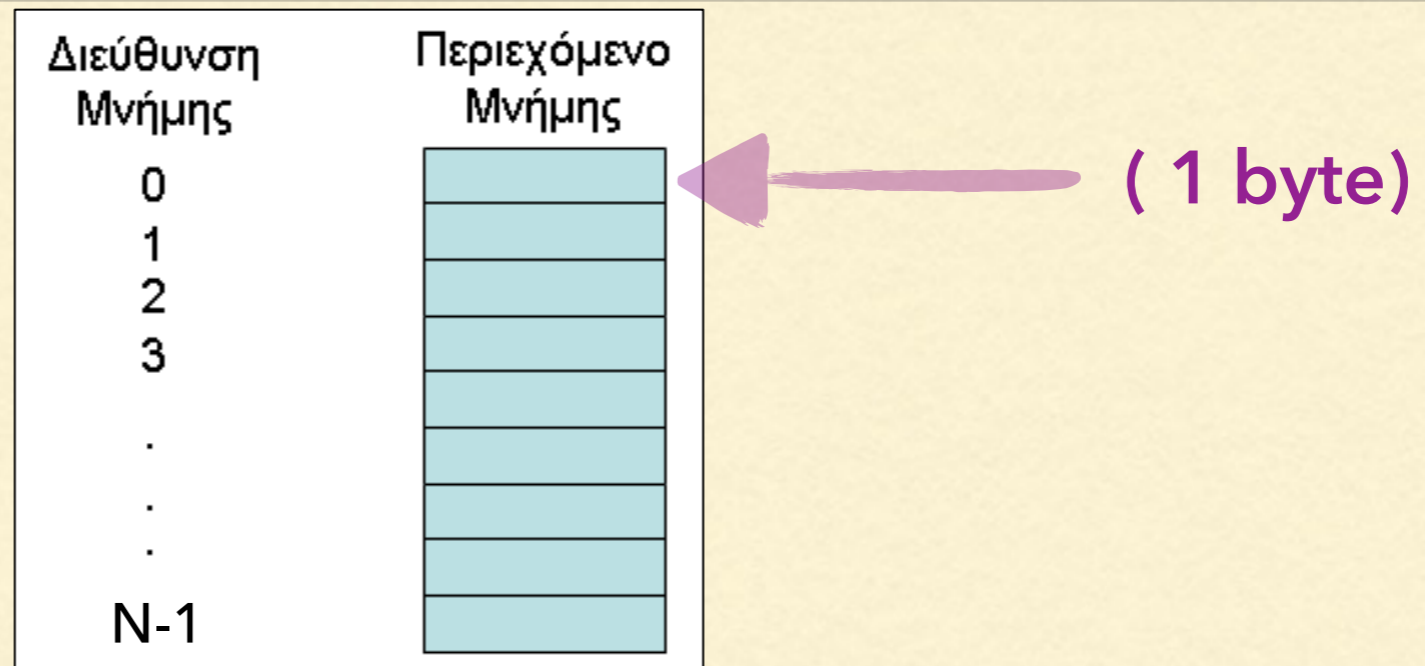
    for(i=1; i<=XSIZE; i++)
    {
        printf("\n");
        for(j=1; j<=YSIZE; j++)
            printf("a[%d][%d] = %d    ", i, j, a[i][j]);
    }

    return 0;
}
```

a[1][1] = 2	a[1][2] = 5	a[1][3] = 10
a[2][1] = 5	a[2][2] = 8	a[2][3] = 13



# ΜΝΗΜΗ ΥΠΟΛΟΓΙΣΤΗ

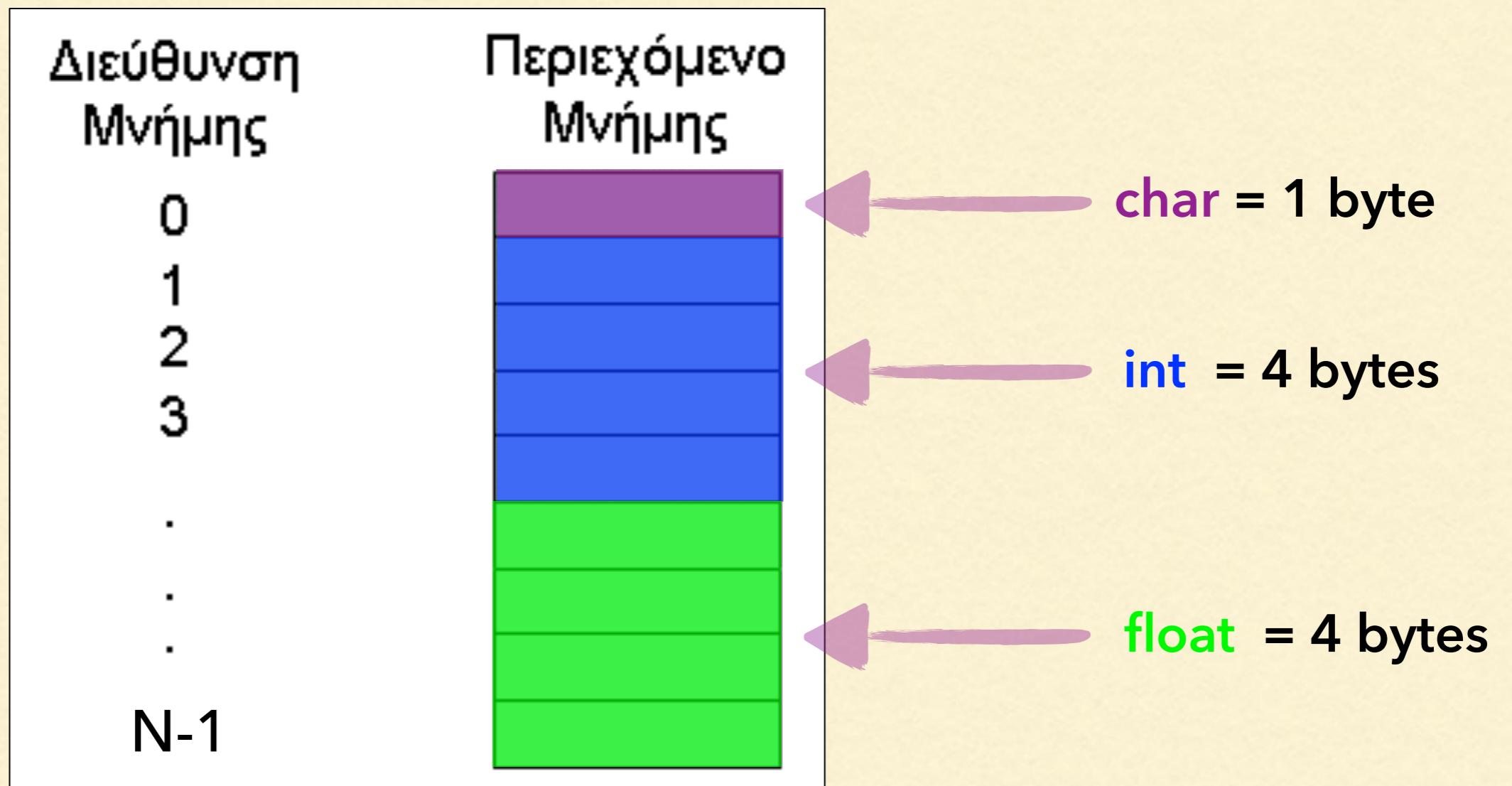


Η μνήμη **RAM** (Random Access Memory) ενός υπολογιστή αποτελείται από μεγάλο πλήθος **N** θέσεων αποθήκευσης δεδομένων με **διαδοχική αρίθμηση**.

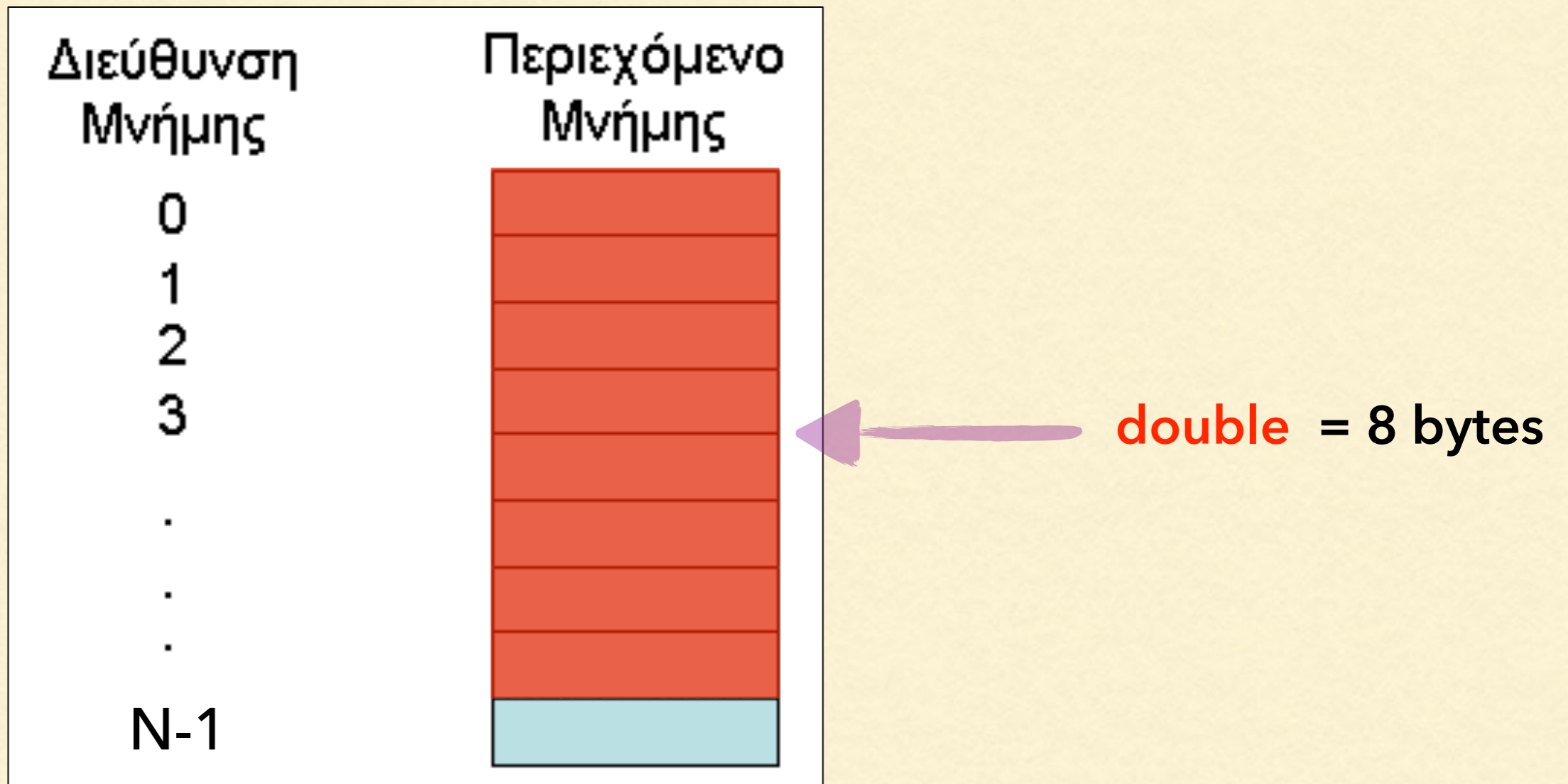
Κάθε θέση ή κελί μνήμης προσδιορίζεται από μία μοναδική **διεύθυνση**, δηλ. από έναν αύξοντα αριθμό με τιμή από 0 έως μία μέγιστη τιμή **N-1**.

Το **περιεχόμενο** της κάθε θέσης μνήμης είναι ένας ακέραιος αριθμός με μέγεθος **1 byte**.

# ΜΝΗΜΗ ΥΠΟΛΟΓΙΣΤΗ

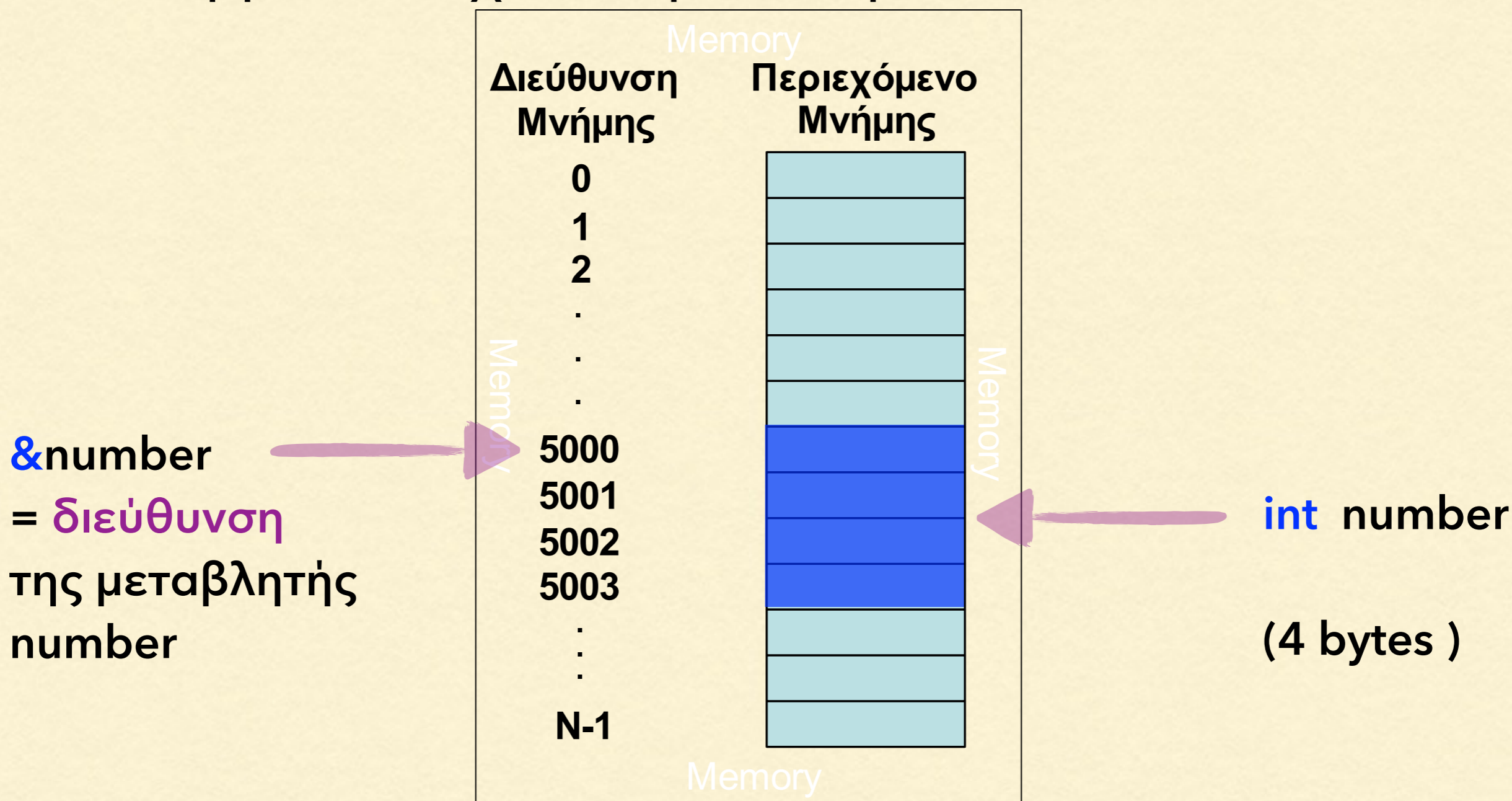


# ΜΝΗΜΗ ΥΠΟΛΟΓΙΣΤΗ



# ΜΝΗΜΗ ΥΠΟΛΟΓΙΣΤΗ

Όταν μια μεταβλητή καταλαμβάνει περισσότερες από μία θέσης μνήμης, τότε ως διεύθυνση της μεταβλητής θεωρείται η διεύθυνση της πρώτης θέσης μνήμης που καταλαμβάνει. Π.χ. αν δηλώσουμε `int number;` τότε:



# ΔΕΙΚΤΗΣ - ΔΙΕΥΘΥΝΣΗ - ΠΕΡΙΕΧΟΜΕΝΟ

---

Μπορούμε να δηλώσουμε αρχικά έναν **ελεύθερο δείκτη** που έχει την ίδια χρησιμότητα όπως ένας σελιδοδείκτης για ένα βιβλίο. Π.χ. δηλώνουμε

```
int *number_pt;
```

Στη συνέχεια, μπορούμε να εκχωρήσουμε στον δείκτη αυτόν τη **διεύθυνση μνήμης** κάποιας συγκεκριμένης **μεταβλητής**, π.χ.

```
number_pt = &a;
```

Αποκτάμε πρόσβαση στο **περιεχόμενο** που βρίσκεται σε αυτή τη διεύθυνση με: **(\*number\_pt)**

# ΠΑΡΑΔΕΙΓΜΑ 1

```
#include <stdio.h>

int main(void)
{
    int *number_pt, a;

    printf("Αρχική τυχαία διεύθυνση: number_pt = %p\n", number_pt);
    printf("Αρχικό τυχαίο περιεχόμενο: (*number_pt) = %d\n", (*number_pt) );

    a = 9;

    printf("\nΔιεύθυνση μεταβλητής: &a = %p\n", &a);
    printf("Τιμή μεταβλητής: a = %d\n", a );

    number_pt = &a;

    printf("\nΝέα διεύθυνση δείκτη: number_pt = %p\n", number_pt);
    printf("Νέο περιεχόμενο (*number_pt) = %d\n", (*number_pt));

    return 0;
}
```

# ΠΑΡΑΔΕΙΓΜΑ 1

---

Αρχική τυχαία διεύθυνση: `number_pt = 0x7fff59b4abd0`

Αρχικό τυχαίο περιεχόμενο: `(*number_pt) = 0`

Διεύθυνση μεταβλητής: `&a = 0x7fff59b4abac`

Τιμή μεταβλητής: `a = 9`

Νέα διεύθυνση δείκτη: `number_pt = 0x7fff59b4abac`

Νέο περιεχόμενο `(*number_pt) = 9`

## ΠΑΡΑΔΕΙΓΜΑ 2

```
#include <stdio.h>

int main(void)
{
    int *temp_pt, a, b;

    a = 9;

    temp_pt = &a;

    b = (*temp_pt);

    printf("a = %d    b = %d\n", a, b);

    return 0;
}
```

a = 9 b = 9



# ΠΑΡΑΔΕΙΓΜΑ 3

```
#include <stdio.h>

int main(void)
{
    int *temp_pt, a, b;

    a = 9;

    temp_pt = &a;

    printf("( *temp_pt ) = %d\n", (*temp_pt));

    a = 31;

    printf("( *temp_pt ) = %d\n", (*temp_pt));

    return 0;
}
```

άλλαξε η τιμή  
της μεταβλητής  
a

άλλαξε αυτομάτως  
το περιεχόμενο  
στη διεύθυνση μνήμης  
που δείχνει ο δείκτης

```
( *temp_pt ) = 9
( *temp_pt ) = 31
```

# ΔΥΝΑΜΙΚΟΙ ΠΙΝΑΚΕΣ

---

Έναν ελεύθερο δείκτη `int *a_pt;` μπορούμε να τον “δείξουμε” προς μια **νέα περιοχή της μνήμης που δημιουργούμε δυναμικά** κατά τη διάρκεια εκτέλεσης ενός προγράμματος. Π.χ. με

```
a_pt = malloc( (N+1)*sizeof(double) );
```

δημιουργούμε μια περιοχή μνήμης για  $N+1$  αριθμούς `double`.

Ουσιαστικά, δημιουργούμε έναν πίνακα `a_pt[N+1]` με το μέγεθος  $N$  να είναι δυναμικό (μπορεί να είναι **ακέραια μεταβλητή** αντί για κάποια σταθερά).

Τα στοιχεία του πίνακα εξακολουθούν να είναι π.χ. `a_pt[i]` (όπως και για πίνακες με σταθερό μέγεθος).

Η συνάρτηση `malloc` (memory allocation) περιέχεται στη βιβλιοθήκη `stdlib.h`.

Στο τέλος πρέπει να ελευθερώσουμε τη μνήμη που δημιουργήσαμε δυναμικά με `free(a_pt)`.

# ΠΑΡΑΔΕΙΓΜΑ

```
#include <stdio.h>
#include <stdlib.h> ( δήλωση νέας βιβλιοθήκης )

int main(void)
{
    int i, N;

    double *a_pt;

    N = 5;

    a_pt = malloc((N+1)*sizeof(double)); ( δήμευση
                                         μνήμης )

    printf("a[%d] = %f\n", 0, a_pt[0]);
    printf("\n");

    for(i=1; i<=N; i++)
    {
        a_pt[i] = i*i;
        printf("a[%d] = %f\n", i, a_pt[i]);
    }

    printf("\na[%d] = %f\n", N+1, a_pt[N+1]);

    free(a_pt); ( απελευθέρωση μνήμης )

    return 0;
}
```

# ΠΑΡΑΔΕΙΓΜΑ

$a[0] = 0.000000$

← ( αγνοούμε το στοιχείο  $a[0]$  )

$a[1] = 1.000000$

$a[2] = 4.000000$

$a[3] = 9.000000$

$a[4] = 16.000000$

$a[5] = 25.000000$

← ( χρήσιμα στοιχεία  
 $a[1]$  έως  $a[N]$  )

$a[6] = 0.000000$

← ( υπέρβαση ορίων δείκτη θέσης! )

# ΔΥΝΑΜΙΚΟΙ ΠΙΝΑΚΕΣ

---

Αν κατά τη διάρκεια εκτέλεσης του προγράμματος προκύψει ανάγκη να μεταβάλλουμε το μέγεθος ενός πίνακα, αυτό γίνεται με την εντολή **realloc** (reallocation). Π.χ. με

```
a_pt = realloc( a_pt, (2*N+1)*sizeof(double) );
```

διπλασιάζουμε το μέγεθος του πίνακα `a_pt` που αρχικά ορίσαμε να έχει μέγεθος `N+1`.

Όταν **μεγαλώνουμε** το μέγεθος ενός πίνακα, τα αρχικά του περιεχόμενα δε διαγράφονται, απλώς προστίθεται επιπλέον χώρος για νέα στοιχεία.

Όταν **μειώνουμε** το μέγεθος ενός πίνακα, απελευθερώνεται μόνο το μέρος της μνήμης που ελαττώνουμε.

# ΠΑΡΑΔΕΙΓΜΑ

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

int main(void)
{
    int i, N;

    double *a_pt;

    N = 5;

    a_pt = malloc((N+1)*sizeof(double));

    for(i=1; i<=N; i++)
        a_pt[i] = i*i;

    a_pt = realloc(a_pt, (2*N+1)*sizeof(double));

    for(i=N+1; i<=2*N; i++)
        a_pt[i] = i*i;

    for(i=1; i<=2*N; i++)
        printf("a[%d] = %f\n", i, a_pt[i]);

    free(a_pt);

    return 0;
}
```

# ΠΑΡΑΔΕΙΓΜΑ

---

```
a[1] = 1.000000  
a[2] = 4.000000  
a[3] = 9.000000  
a[4] = 16.000000  
a[5] = 25.000000  
a[6] = 36.000000  
a[7] = 49.000000  
a[8] = 64.000000  
a[9] = 81.000000  
a[10] = 100.000000
```