

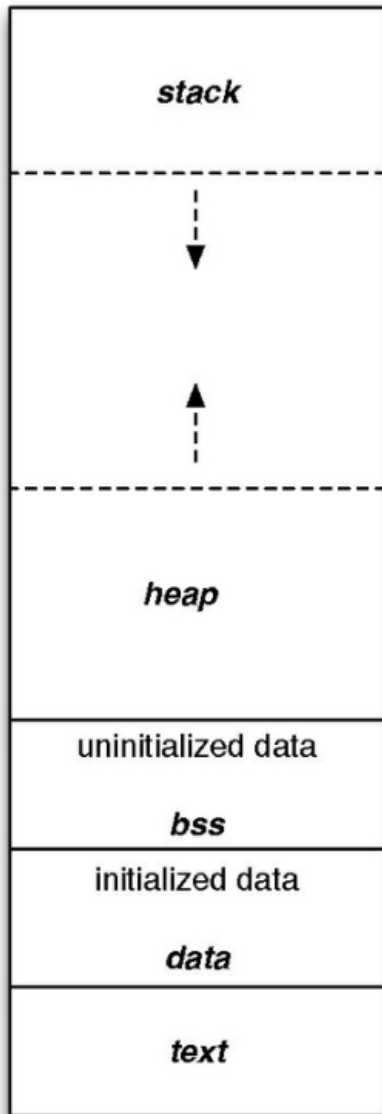


**Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών**

Εισαγωγή στον Προγραμματισμό

Η γλώσσα προγραμματισμού C

Memory Layout of a C program

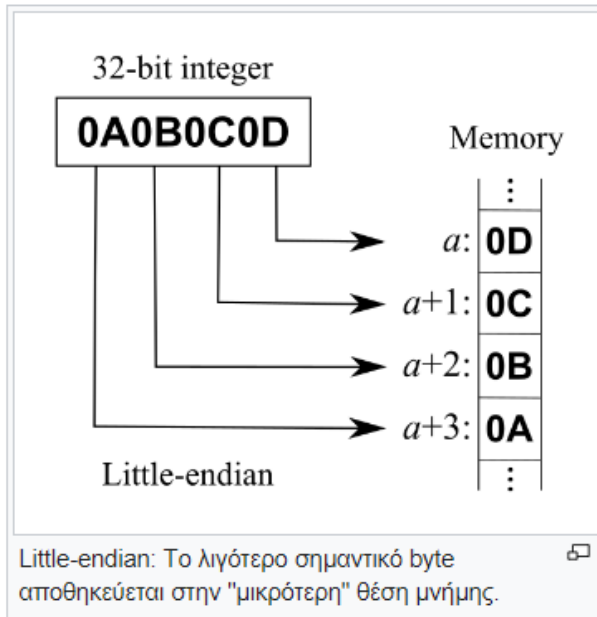


Διεύθυνση μεταβλητής

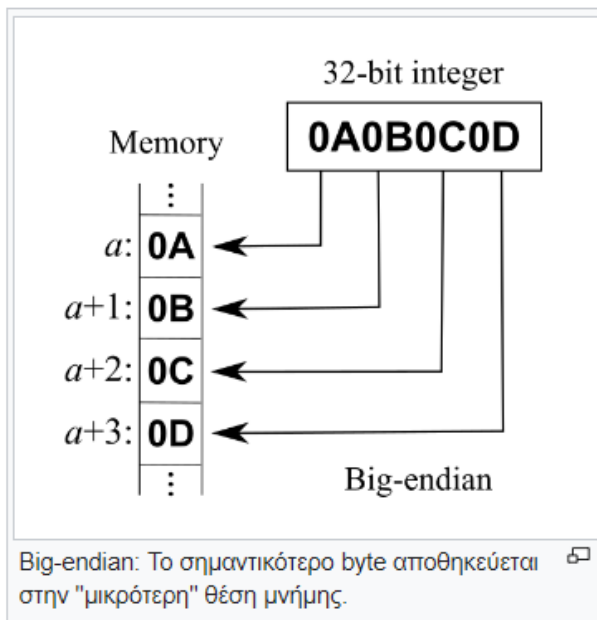
- Κάθε μεταβλητή καταλαμβάνει ένα ή περισσότερα byte στη μνήμη
 - char (1B), short (2B), int (4B), long (8B), float (4B), double (8B), κ.τ.λ.
 - Η διεύθυνση του πρώτου byte είναι η **διεύθυνση της μεταβλητής**

Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00	sum	int (4 bytes)	000000FF (255 ₁₀)
90000001	00			
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF (-1 ₁₀)
90000005	FF			
90000006	1F			
90000007	FF	average	double (8 bytes)	1FFFFFFFFFFFFFFF (4.45015E-308 ₁₀)
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90	ptrSum	int* (4 bytes)	90000000
9000000F	00			
90000010	00			
90000011	00			

Note: All numbers in hexadecimal



<https://en.wikipedia.org/wiki/Endianness>



Διεύθυνση μεταβλητής

- Κάθε μεταβλητή καταλαμβάνει ένα ή περισσότερα byte στη μνήμη
 - **char** (1B), **short** (2B), **int** (4B), **long** (8B), **float** (4B), **double** (8B), κ.τ.λ.
 - Η διεύθυνση του πρώτου byte είναι η **διεύθυνση της μεταβλητής**

Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00	sum	int (4 bytes)	000000FF (255 ₁₀)
90000001	00			
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF (-1 ₁₀)
90000005	FF			
90000006	1F	average	double (8 bytes)	1FFFFFFFFFFFFFFF (4.45015E-308 ₁₀)
90000007	FF			
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF	ptrSum	int* (4 bytes)	90000000
9000000D	FF			
9000000E	90			
9000000F	00			
90000010	00			
90000011	00			

Note: All numbers in hexadecimal

Δείκτες

- Η C δίνει τη δυνατότητα να αποθηκεύουμε **διευθύνσεις μνήμης** σε ειδικές μεταβλητές που λέγονται **pointers**

```
int *i;      // pointer to int
char *c;     // pointer to char
double *d;   // pointer to double
```

- Ο κάθε pointer είναι associated με κάποιον τύπο δεδομένων.
 - Π.χ., το *i* στο παραπάνω παράδειγμα δείχνει σε int
- Ορίζονται με ένα αστεράκι πριν το όνομα της μεταβλητής
 - Π.χ., στο παρακάτω, μόνο το *p* είναι pointer

```
int i, a[100], *p;
```

Τελεστές Δεικτών

□ Τελεστής διεύθυνσης (address operator): **&**

- Επιστρέφει τη διεύθυνση μιας μεταβλητής

```
int i, *p;  
p = &i;    // p = address of i
```

□ Τελεστής αποαναφοράς (dereference operator): *****

- Μας επιτρέπει να προσπελάσουμε το περιεχόμενο της μνήμης που δείχνει ένας pointer

```
// συνέχεια από το προηγούμενο  
*p = 42;  
printf("%d\n", i);    // 42  
i /= 2;  
printf("%d\n", *p);    // 21
```

Τελεστές Δεικτών

□ Τελεστής ανάθεσης: =

```
int i=5, *p, *q;  
p = &i;    // p = address of i  
q = p;  
printf("%d\n", *q); // 5
```

□ Προσοχή! Άλλο η ανάθεση δείκτη, κι άλλο η ανάθεση της τιμής που δείχνει ο δείκτης

```
q = p;    // Ανάθεση δείκτη  
*q = *p;  // Ανάθεση τιμής
```


Παράδειγμα

```
int x=1;
int y=2;
int *p;
p = &x;  /* η p δείχνει τη διεύθυνση της x */
y = *p;  /* η y γίνεται 1, δηλαδή η τιμή της x */
*p = 0;  /* η x γίνεται 0 */
*p = *p + 10;  /* x=x+10 */

*p += 1;  /* x=x+1 */
++*p;     /* x=x+1 */
(*p)++;   /* x=x+1 */
```

Παράδειγμα

```
#include <stdio.h>

main()
{
    int x=5; int y=10; int *ptr; int **ptr2;
    ptr=&x;
    ptr2=&ptr;
    *ptr2=&y;
    printf("%d", *ptr);
}
```

Τελεστές Δεικτών

- Οι τελεστές **&** και ***** είναι «αντίστροφοι», συνεπώς εν γένει αλληλοαναιρούμενοι

```
int i, j, *p=NULL, *q=&i;  
i = *&j;    // i=j  
i = &*j;    // compile error! (j not ptr)  
p = &*q;    // p=q
```

- Γενικά είναι καλό να αρχικοποιείτε πάντα τους pointers (π.χ., με NULL), αλλιώς μπορεί να «δείχνουν» σε αυθαίρετη διεύθυνση με ολέθρια αποτελέσματα!
 - Τι είναι το NULL???
 - Είναι ειδική τιμή (συγκεκριμένα η τιμή 0), που σημαίνει τον **κενό δείκτη**, δηλαδή έναν pointer που δεν δείχνει πουθενά
 - Ο NULL pointer δεν αποδεικτοδοτείται. Π.χ., το `p=NULL`; `*p = 5`; θα δημιουργήσει λάθος, δεν θα βάλει το 5 στη διεύθυνση 0.

NULL

□ Τι είναι το **NULL**???

- Είναι ειδική τιμή (συγκεκριμένα η τιμή 0), που σημαίνει τον **κενό δείκτη**, δηλαδή έναν pointer που δεν δείχνει πουθενά
- Ο NULL pointer δεν αποδεικτοδοτείται
- Π.χ., ο παρακάτω κώδικας θα δημιουργήσει λάθος, δεν θα βάλει το 5 στη διεύθυνση 0

```
int *p = NULL;  
*p = 5;
```

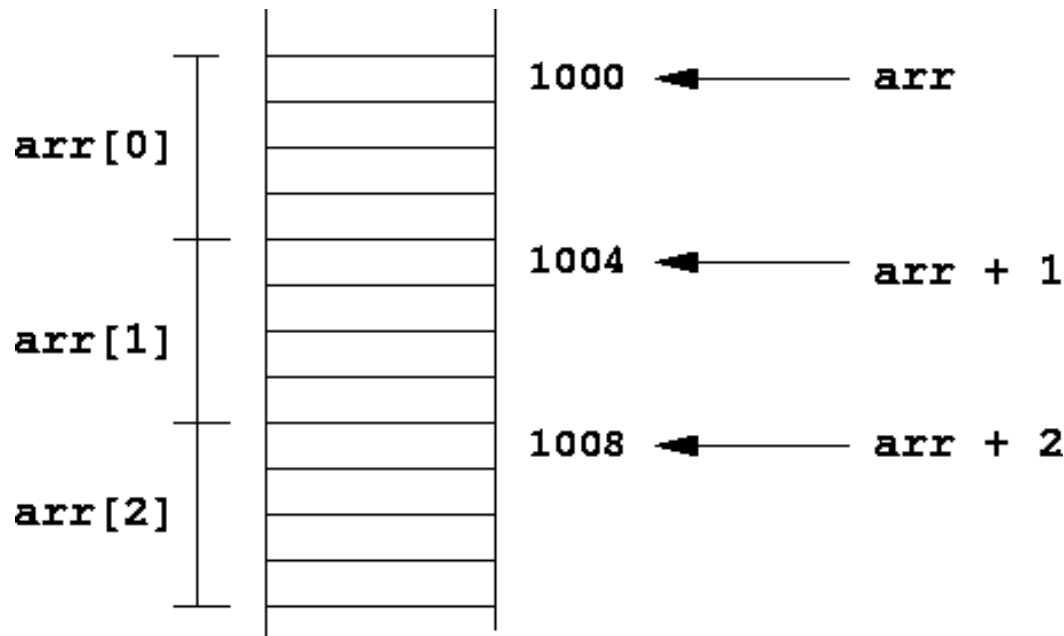
□ Και τι είναι το **void ***???

- Είναι ο τύπος δεδομένων γενικού δείκτη (δείκτη σε κενό)
- Δηλαδή δείκτης που δεν είναι associated με συγκεκριμένο τύπο (int, char, κ.τ.λ.)
- Κανονικός δείκτης, απλά δεν μπορεί να κάνει **pointer arithmetic**
- Μα, τι είναι pointer arithmetic??? 😊

Pointer Arithmetic (Αριθμητική Δεικτών)

- Ας θεωρήσουμε ότι έχουμε ένα array ακεραίων **int arr[10]**
- Γνωρίζουμε πως **arr[i]** είναι το i-οστό στοιχείο του array
- Αυτό που δεν γνωρίζουμε είναι πως το **arr[i]** ορίζεται ως ***(arr + i)**

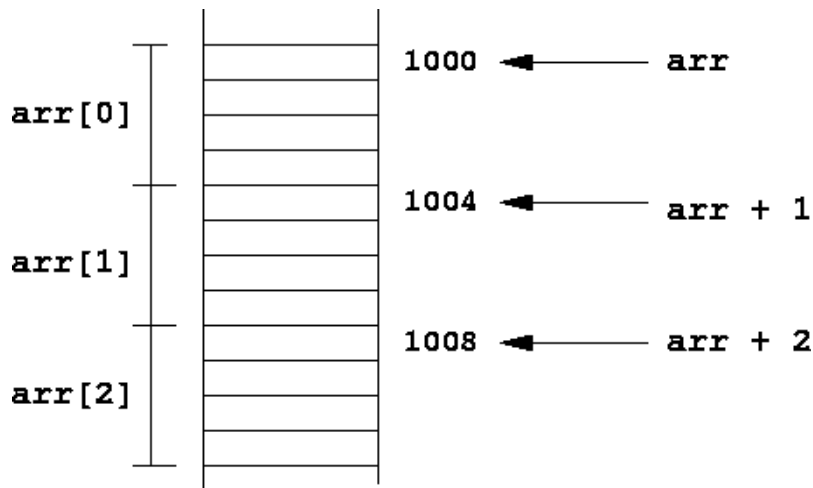
```
arr == &arr[0]  
arr[0] == *arr  
arr[i] == *(arr + i)
```



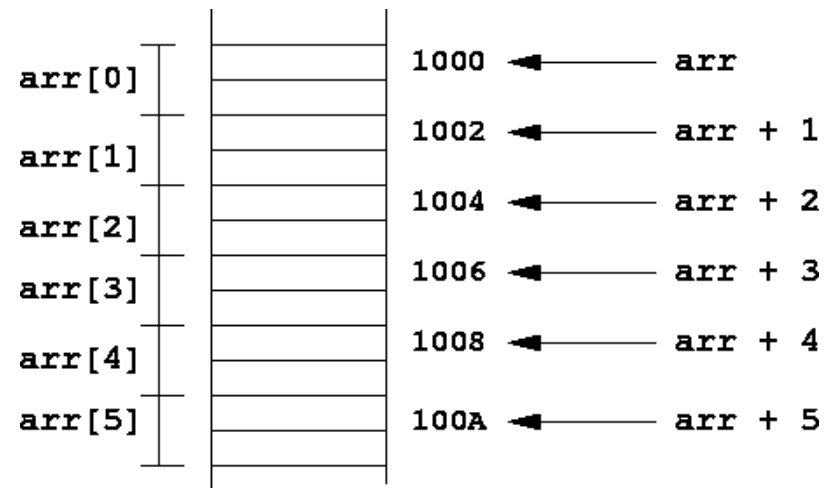
Pointer Arithmetic (Αριθμητική Δεικτών)

Το κατά πόσο αυξάνεται/μειώνεται ένας pointer σε πράξεις με ακεραίους, εξαρτάται από τον τύπο του.

int arr[10];
sizeof(int) == 4



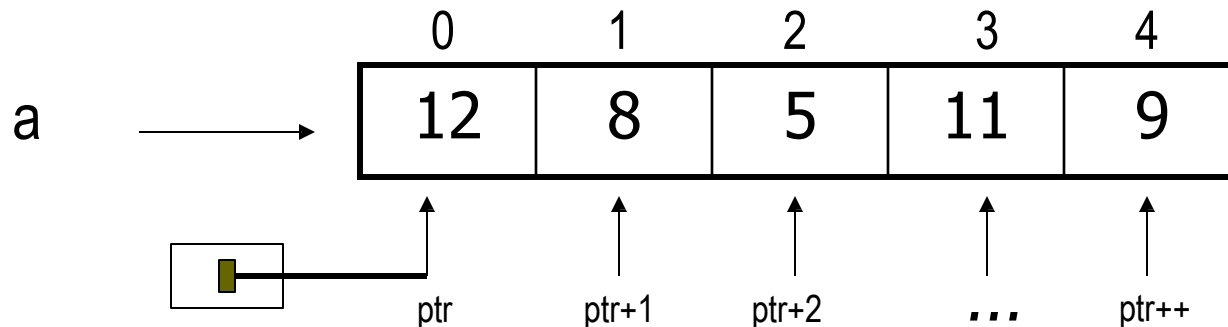
short arr[10];
sizeof(short) == 2



Pointers και Arrays

- Μπορούμε να δώσουμε σε έναν pointer την διεύθυνση ενός array, και στη συνέχεια να τον χρησιμοποιήσουμε ως array

```
int a[] = {12, 8, 5, 11, 9}, *ptr;  
ptr=a;  
printf("%d\n", ptr[1]);    // 8  
ptr += 2;  
printf("%d\n", ptr-a);    // 2  
printf("%d\n", ptr[1]);    // 11  
ptr--;  
printf("%d\n", ptr[1]);    // 5  
a++;    // Compile error, arr is const!
```



Pointer Arithmetic

- Έγκυρες πράξεις με δείκτες είναι:
 - απόδοση τιμής με δείκτη ίδιου τύπου
 - η πρόσθεση ή αφαίρεση δείκτη και ακεραίου
 - η αφαίρεση ή σύγκριση δύο δεικτών για μέλη ίδιου πίνακα
 - η αντικατάσταση ή σύγκριση με NULL

- Δεν είναι έγκυρες πράξεις οι:
 - πρόσθεση δύο δεικτών
 - πολλαπλασιασμός, διαίρεση, ολίσθηση, η πρόσθεση float, double σε δείκτες.

Πέρασμα Μεταβλητών

```
#include <stdio.h>
void swap(int x, int y);

int main()
{
    int x=2, y=3;
    printf("x:%d, y:%d \n", x,y);
    swap(x,y);
    printf("x:%d, y:%d \n", x,y);
}

void swap(int x, int y)  /* λάθος */
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

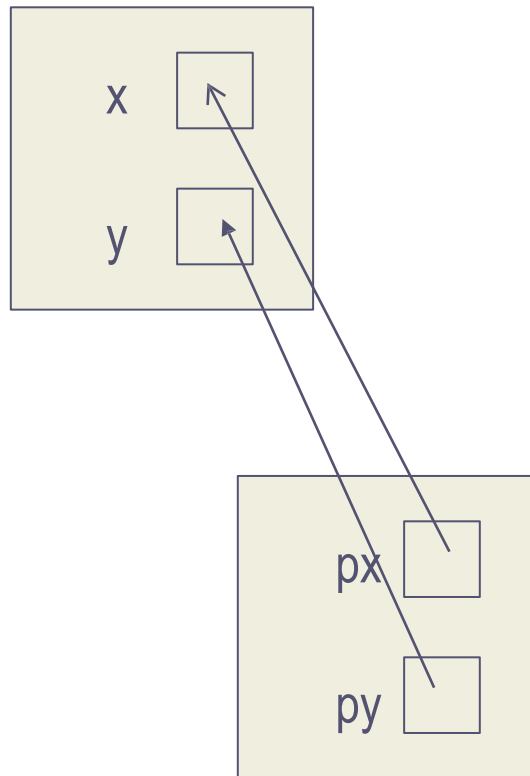
Πέρασμα Μεταβλητών (2)

```
#include <stdio.h>
void swap(int x, int y);

int main()
{
    int x=2, y=3;
    printf("x:%d, y:%d \n", x,y);
    swap(&x,&y);
    printf("x:%d, y:%d \n", x,y);
}

void swap(int *px, int *py) /*σωστή υλοποίηση
*/
{
    int temp;
    temp=*px;
    *px=*py;
    *py=temp;
}
```

Πέρασμα Μεταβλητών (3)



Δείκτες - Αλφαριθμητικά (1)

Δήλωση δείκτη αλφαριθμητικού

char * <όνομα – δείκτη>;

π.χ. char * pmsg;

Ανάθεση σε δείκτη αλφαριθμητικού

<όνομα-δείκτη> = < αλφαριθμητικό >;

π.χ. pmsg = "Today is Thursday";

Δείκτες - Αλφαριθμητικά (2)

Προσοχή στις διαφορές

`char msg[] = "Today is Thursday";` (πίνακας χαρακτήρων)

`char * pmsg = "Today is Thursday";` (δείκτης σε πίνακα, μπορεί να πάρει άλλη τιμή)

`char * msg[18];` (πίνακας δεικτών χαρακτήρα)

`msg[1]` -> 2ος δείκτης

`* (msg[1])` -> ο 1ος χαρακτήρας του δεύτερου δείκτη

Παράδειγμα (2)

```
void strcpy(char *s, char *t)
{
    int i=0;
    while ((s[i]=t[i])!='\0')
        i++;
}
```

```
void strcpy(char *s, char *t)
{
    while ((*s=*t) != '\0') { s++; t++;}
}
```

```
void strcpy(char *s, char *t)
{
    while ((* s++=*t++) != '\0') ;
}
```

Παράδειγμα (3)

```
int strcmp(char *s, char *t)
{
    int i=0;
    for (i=0; (s[i]==t[i])&& s[i]!='\0'; i++);
    return s[i]-t[i];
}
```

```
int strcmp(char *s, char *t)
{
    for (; (*s==*t) && *s!='\0'; s++, t++);
    return *s-*t;
}
```

Ορίστε μία συνάρτηση που παίρνει σαν όρισμα ένα αλφαριθμητικό και μετατρέπει το αλφαριθμητικό από μικρά γράμματα σε κεφαλαία..

Θεωρήστε σαν δεδομένο την ύπαρξη των συναρτήσεων:

```
int islower(int c)
```

```
int toupper(int c)
```


Δυναμική Διαχείριση Μνήμης

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

- ***Calloc, Malloc,***
για δέσμευση μνήμης
- ***Realloc***
για αλλαγή μεγέθους δεσμευμένης μνήμης
- ***Free***
για απελευθέρωση μνήμης

Malloc()

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

```
void *malloc(size_t size);
```

Η συνάρτηση malloc δεσμεύει χώρο μεγέθους τουλάχιστον size bytes.

Επιστρέφει ένα δείκτη στη δεσμευθείσα μνήμη ή NULL, αν δεν υπάρχει

διαθέσιμη μνήμη. Τα bytes του χώρου μνήμης δεν παίρνουν αρχική τιμή

```
x=(int *)malloc(n*sizeof(int)); // int *x; (σαν int x[n])
```

```
x=(int **)malloc(n*sizeof(int *)); //int **x; (σαν int x[n][m])
```

```
for (i=0; i<n; i++) *(x+i)=(int *)malloc(m*sizeof(int));
```

```
&x[i][j] ---- *(x+i)+j
```

```
x[i][j] ---- *(* (x+i)+j)
```

Calloc()

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

```
void *calloc(size_t n, size_t size);
```

Η συνάρτηση calloc δεσμεύει χώρο για ένα πίνακα n στοιχείων, μεγέθους size το καθένα. Διασφαλίζει την αρχικοποίησή της με 0.

Επιστρέφει ένα δείκτη στη δεσμευθείσα μνήμη ή NULL, αν δεν υπάρχει διαθέσιμη μνήμη.

```
p=(int *)calloc(n, sizeof(int));
```

Realloc()

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

```
void *realloc(void *buffer, size_t size);
```

Η συνάρτηση `realloc` μεταβάλλει το μέγεθος ενός τμήματος μνήμης που είχε προηγουμένα δεσμευτεί. Το νέο μέγεθος ορίζεται από τη `size`. Διασφαλίζει τα υπάρχοντα περιεχόμενα στη μνήμη. Επιστρέφει ένα δείκτη στο νέο τμήμα μνήμης που μπορεί να είναι ίδιος με τον `buffer` ή διαφορετικός αν το τμήμα μετακινηθεί. Αν ο `buffer` είναι `NULL`, η `realloc` λειτουργεί σαν τη `malloc`.

```
p=(int *) realloc(p, n*sizeof(int));
```

Free ()

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

```
void free(void * buffer);
```

Η συνάρτηση `free` αποδεσμεύει τη μνήμη η οποία σηματοδοτείται από το όρισμα `buffer` και η οποία είχε προηγουμένα δεσμευτεί με κλήση μίας εκ των *calloc*, *malloc*, *realloc*.

Δείκτες – Πίνακες (1)

- Το όνομα ενός πίνακα είναι μία «σταθερά τύπου δείκτη» με τιμή τη διεύθυνση του πρώτου στοιχείου του πίνακα.
- Λόγω αυτού του γεγονότος μπορούμε να ορίσουμε ένα δείκτη στον πίνακα και να χειριστούμε τον πίνακα μέσω αυτού του δείκτη.
- Δεν είναι εφικτή η ανάθεση τιμής και η αριθμητική με δείκτες

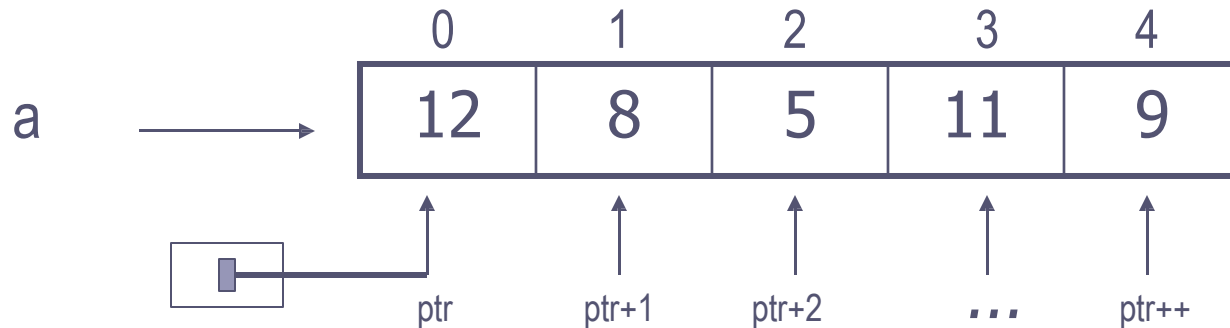
Δείκτες – Πίνακες (2)

Έστω

```
int a[5] = { 12, 8, 5, 11, 9};
```

```
int *ptr;
```

```
ptr = a; ή ptr = &a[0];
```



Προσοχή: `a++`; `a--` (το όνομα του πίνακα δεν είναι δείκτης (δηλ. μεταβλητή δείκτη))

Δείκτες – Πίνακες (3)

Ισοδύναμες εκφράσεις

$*(a+i) \longleftrightarrow a[i]$

$a+i \longleftrightarrow \&a[i]$

Δείκτης σε δείκτη

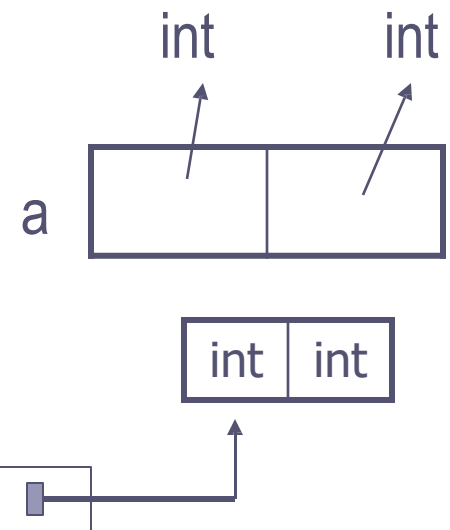
Προτεραιότητα [] μεγαλύτερη από *

π.χ. `int *a[2]`

πίνακας (δύο) δεικτών

`int (*ptr)[2]`

δείκτης σε πίνακα (δύο)
ακεραίων



Πίνακες Πολλών Διαστάσεων & Πολλαπλοί Δείκτες στη C

Σχέση δισδιάστατου πίνακα με διπλό
δείκτη

Τι είναι δισδιάστατος πίνακας στη C;

- Παράδειγμα: `int a[3][4];`
- Τύπος: πίνακας 3 στοιχείων, κάθε στοιχείο πίνακας 4 `int`
- Αποθήκευση: ένα συνεχόμενο μπλοκ μνήμης 3×4 `int`
- `a[i][j] == *(*(a + i) + j)`

Δισδιάστατος πίνακας ως παράμετρος

- Ο πίνακας 'υποβιβάζεται' σε δείκτη στον πρώτο εσωτερικό πίνακα
- `void f(int a[3][4]);`
- ισοδύναμο με: `void f(int a[][4]);`
- ισοδύναμο με: `void f(int (*a)[4]);`

Τι είναι διπλός δείκτης;

- `int **p;` \Rightarrow δείκτης σε δείκτη σε `int`
- Συχνή χρήση ως «δυναμικός 2D πίνακας»
- Πίνακας από δείκτες σε σειρές (γραμμές) `int`

Δυναμική δέσμευση με `int **`

- `int **p;`
- `p = malloc(rows * sizeof *p);` // πίνακας δεικτών
- `for (i = 0; i < rows; i++)`
 `p[i] = malloc(cols * sizeof *p[i]);`
- Κάθε γραμμή μπορεί να βρίσκεται σε διαφορετική θέση στη μνήμη

Συνηθισμένο λάθος: `int[][]` vs `int **`

- `void f(int **a);`
- `int a[3][4];`
- `f(a);` // ΛΑΘΟΣ – ασύμβατοι τύποι
- Ο `a` γίνεται `int (*)[4]`, όχι `int **`
- Καμία εγγύηση συμβατής διάταξης μνήμης → `undefined behavior`

Σωστό πέρασμα 2D πίνακα σε συνάρτηση

- `void f(int a[][4]);`
- `// ή`
- `void f(int (*a)[4]);`
- Χρήση μέσα στη f: `a[i][j]`

ΑΛΛΙΩΣ ΧΡΗΣΗ ΔΙΠΛΟΥ ΔΕΙΚΤΗ

Εναλλακτικός δυναμικός 2D πίνακας (1D μπλοκ)

- `int *p = malloc(rows * cols * sizeof *p);`
- Πρόσβαση στοιχείων: `p[i * cols + j];`
- Ενιαίο, συνεχόμενο μπλοκ μνήμης (όπως ο στατικός 2D πίνακας)

Σύνοψη

- Δισδιάστατος πίνακας \neq διπλός δείκτης (`int **`).
- Κοινή σύνταξη `a[i][j]`, διαφορετικοί τύποι και διάταξη μνήμης.
- 2D πίνακας: `int (*)[cols]`, συνεχόμενη μνήμη.
- `int **`: πίνακας από δείκτες, σειρές πιθανώς διάσπαρτες.
- Δεν περνάμε `int[ROWS][COLS]` σε συνάρτηση που ζητά `int **`.

Ασκήσεις

1. Δημιουργήστε κώδικα ο οποίος χειρίζεται δύο μονοδιάστατους πίνακες ακεραίων αριθμών ίδιου μεγέθους, και στη συνέχεια προσθέτει τα περιεχόμενα τους ανά δύο και τα τοποθετεί στον πρώτο πίνακα. Τό μέγεθος των πινάκων θα πρέπει να δίνεται από τον χρήστη ενώ η ανάγνωση δεδομένων και η πρόσθεση των στοιχείων των πινάκων θα πρέπει να γίνεται με κλήση δύο κατάλληλα ορισμένων συναρτήσεων.

2. Δημιουργήστε κώδικα που χειρίζεται ένα πίνακα δύο διαστάσεων οριζόμενο με τη μορφή δείκτη σε δείκτη, διαβάζει τα περιεχόμενα του πίνακα και στη συνέχεια τα εκτυπώνει. Οι διαστάσεις του πίνακα θα πρέπει να δίνονται από τον χρήστη, η ανάγνωση των περιεχομένων και η εκτύπωση θα πρέπει να γίνεται με κλήση συναρτήσεων.

3. Δημιουργήστε κώδικα ο οποίος χειρίζεται δύο δισδιάστατους πίνακες ακεραίων αριθμών A και B μεγέθους $n \times k$ και $k \times m$ αντίστοιχα (τα n , m , k δίνονται από τον χρήστη). Το πρόγραμμα θα πρέπει να διαβάζει τα περιεχόμενα των δύο πινάκων και στη συνέχεια να αποθηκεύει το γινόμενό τους σε ένα πίνακα C διαστάσεων $n \times m$. Η ανάγνωση των περιεχομένων και ο πολλαπλασιασμός θα πρέπει να γίνεται με την κλήση κατάλληλων συναρτήσεων.