

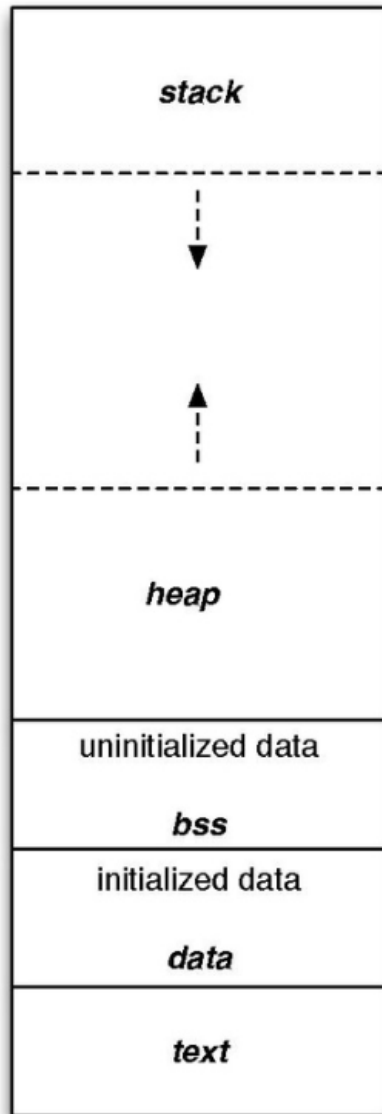


**Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών**

Εισαγωγή στον Προγραμματισμό

Η γλώσσα προγραμματισμού C

Memory Layout of a C program



https://commons.wikimedia.org/wiki/File:Program_memory_layout.pdf

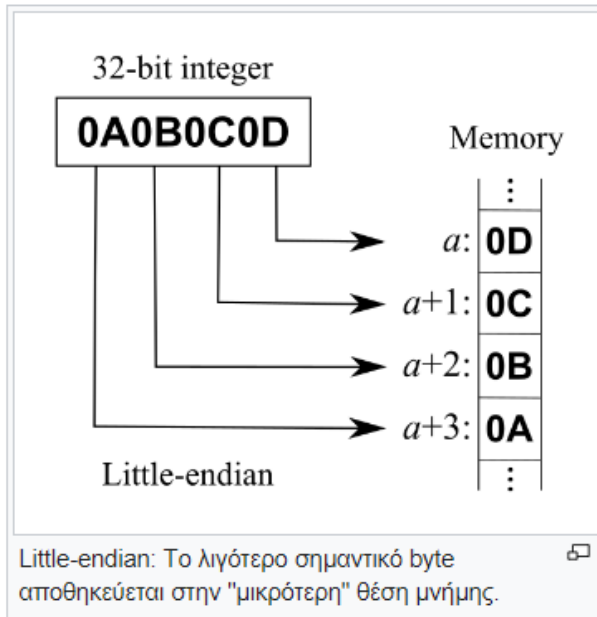
This file is licensed under the [Creative Commons Attribution-Share Alike 3.0 Unported](https://creativecommons.org/licenses/by-sa/3.0/) license.

Διεύθυνση μεταβλητής

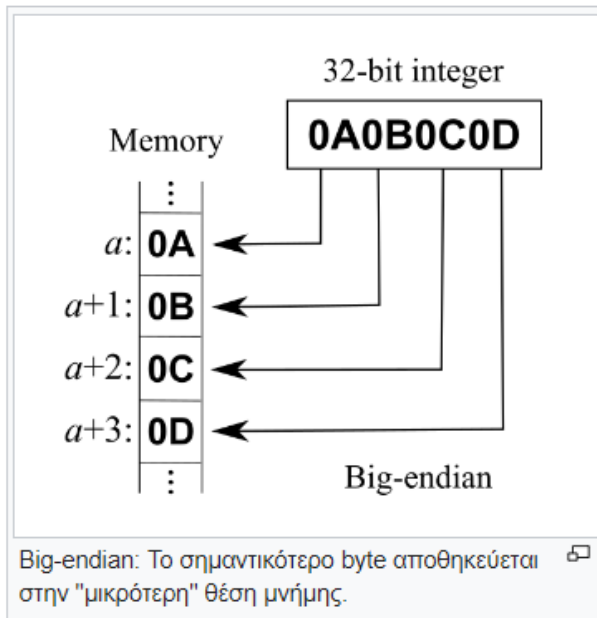
- Κάθε μεταβλητή καταλαμβάνει ένα ή περισσότερα byte στη μνήμη
 - char (1B), short (2B), int (4B), long (8B), float (4B), double (8B), κ.τ.λ.
 - Η διεύθυνση του πρώτου byte είναι η **διεύθυνση της μεταβλητής**

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			
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			
9000000D	FF			
9000000E	90	ptrSum	int* (4 bytes)	90000000
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

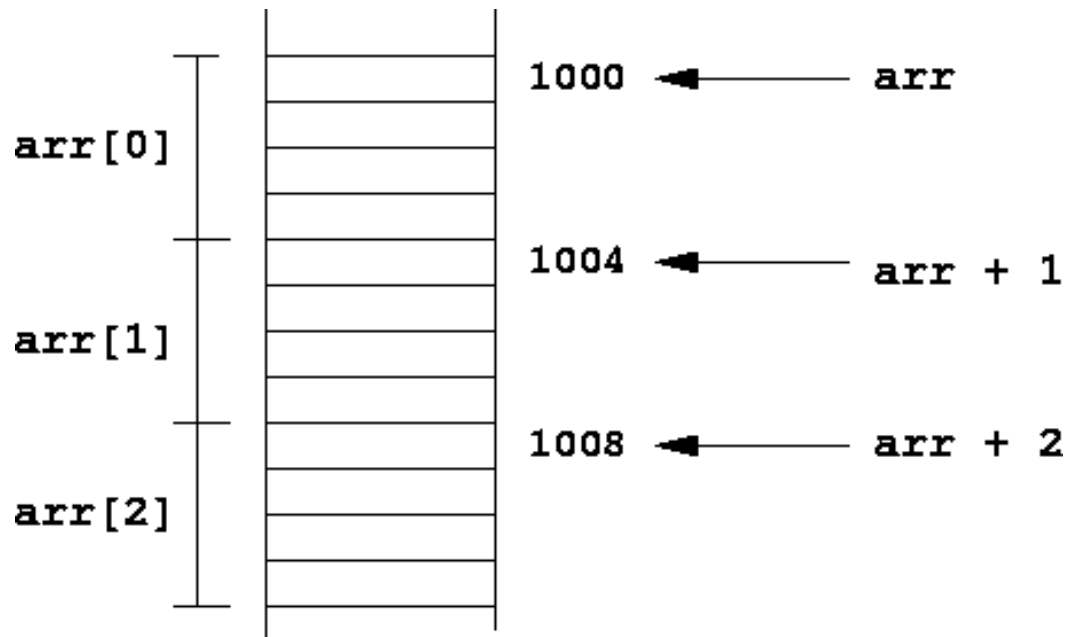
- Έγκυρες πράξεις με δείκτες είναι:
 - απόδοση τιμής με δείκτη ίδιου τύπου
 - η πρόσθεση ή αφαίρεση δείκτη και ακεραίου
 - η αφαίρεση ή σύγκριση δύο δεικτών για μέλη ίδιου πίνακα
 - η αντικατάσταση ή σύγκριση με NULL

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

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)**

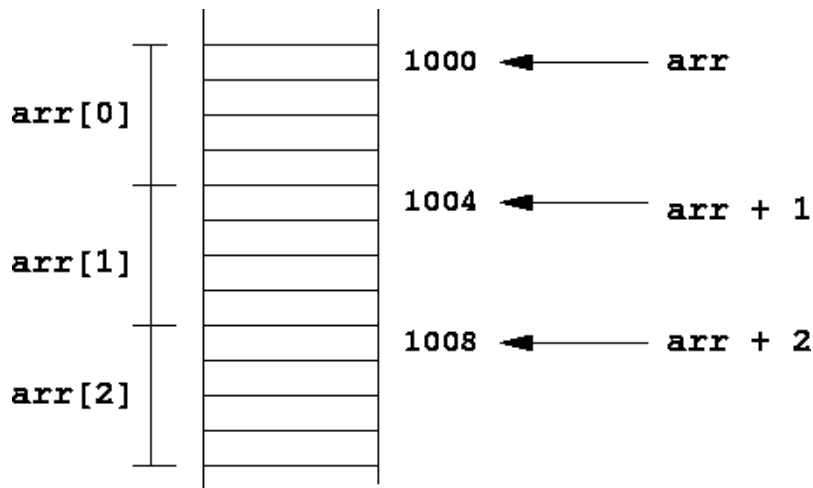
```
arr == &arr[0]  
arr[0] == *arr  
arr[i] == *(arr + i)
```



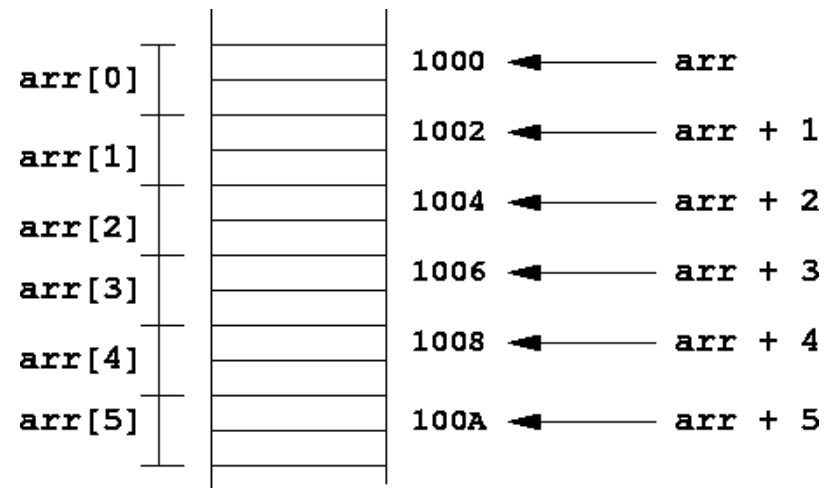
Pointer Arithmetic (Αριθμητική Δεικτών)

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

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



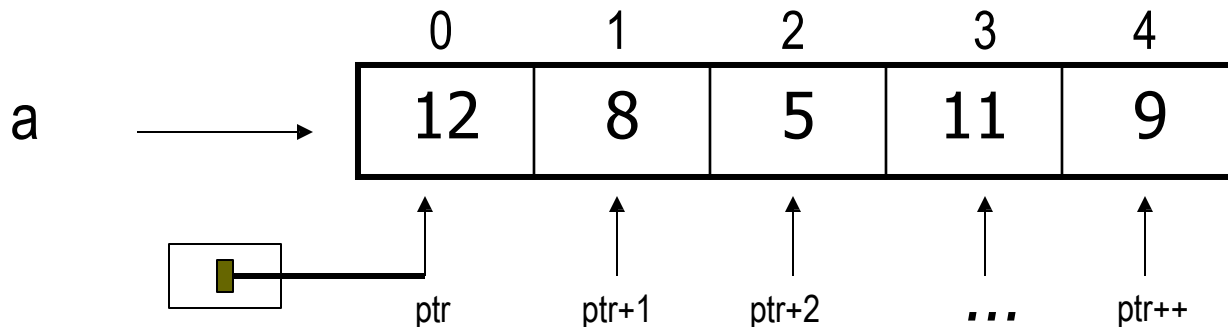
short arr[10];
sizeof(short) == 2



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!
```



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

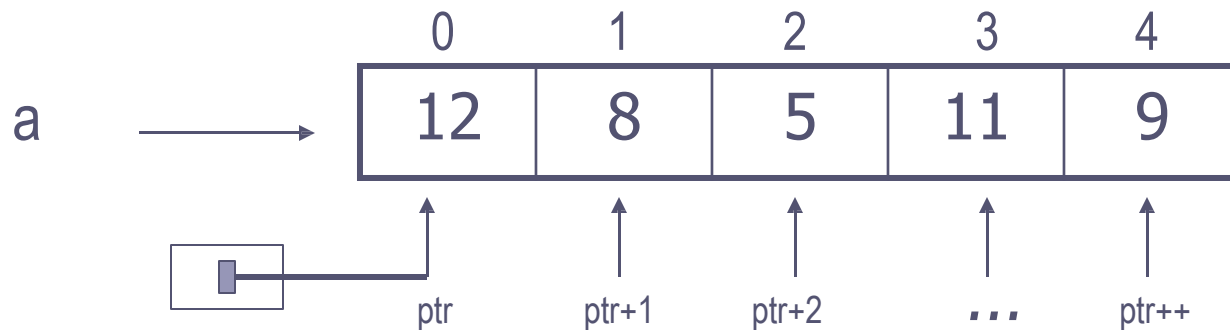
Δείκτες – Πίνακες (2)

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

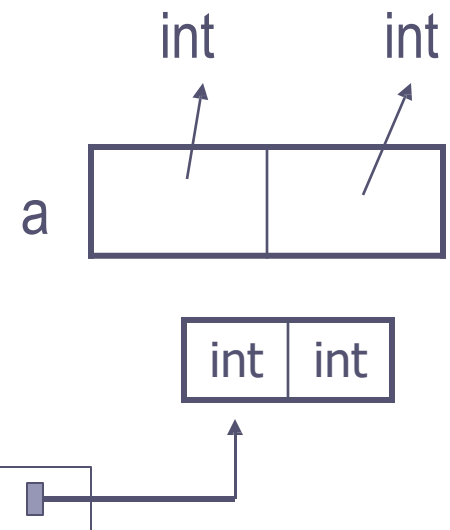
Προτεραιότητα [] μεγαλύτερη από *

π.χ. `int *a[2]`

πίνακας (δύο) δεικτών

`int (*ptr)[2]`

δείκτης σε πίνακα (δύο)
ακεραίων



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;
}
```

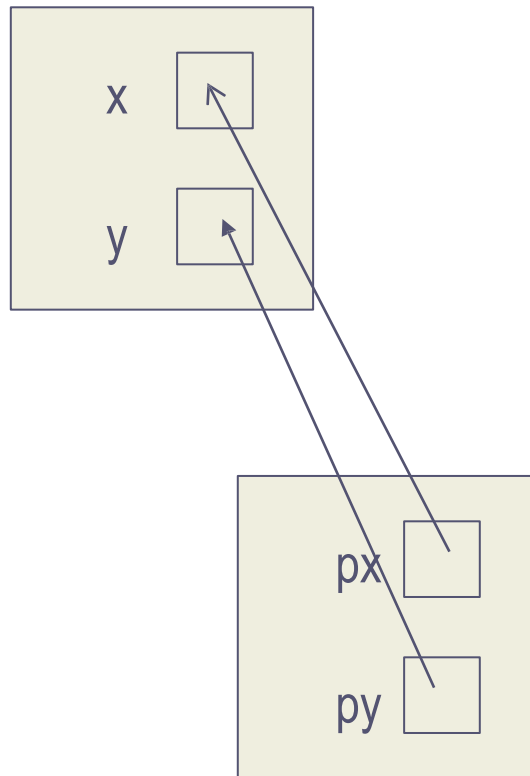
Πέρασμα Μεταβλητών (2)

```
#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)



Δυναμική Διαχείριση Μνήμης

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

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