Η γλώσσα C

Δείκτες (pointers)

Μεταβλητές

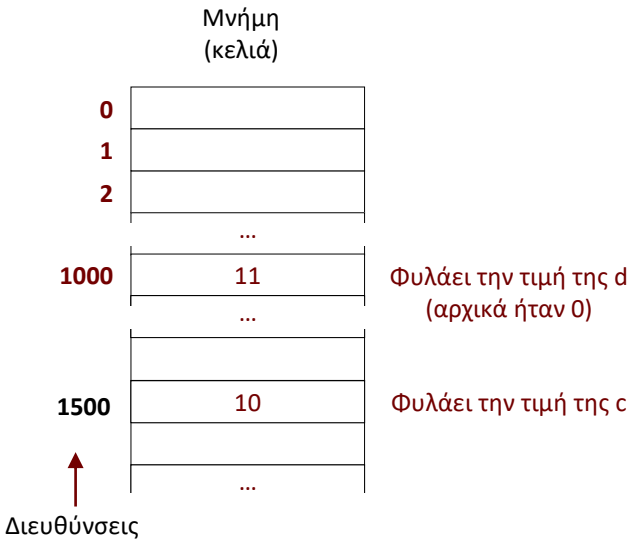
- Οι μεταβλητές πρώτα **δηλώνονται** και έπειτα **χρησιμοποιούνται** σε εκφράσεις/εντολές
- Όταν δηλώνονται, *ονοματίζονται*
- Όταν χρησιμοποιούνται, *αναφέρουμε το όνομά τους* και
  - στην έκφραση χρησιμοποιείται η τιμή τους
  - Όταν είναι στο αριστερό μέρος μιας καταχώρησης, τροποποιείται η τιμή τους

```
#include <stdio.h>

int main() {
  int c=10, d=0;      /* δήλωση  */

  printf("%d", 2*c); /* έκφραση */
  d = c + 1;         /* έκφραση */
}
```

- Η τιμή των μεταβλητών αποθηκεύεται σε κελιά στην μνήμη
  - Το όνομά τους *δεν αποθηκεύεται πουθενά!*
  - Το κελί που φυλάει την τιμή ονομάζεται **διεύθυνση**



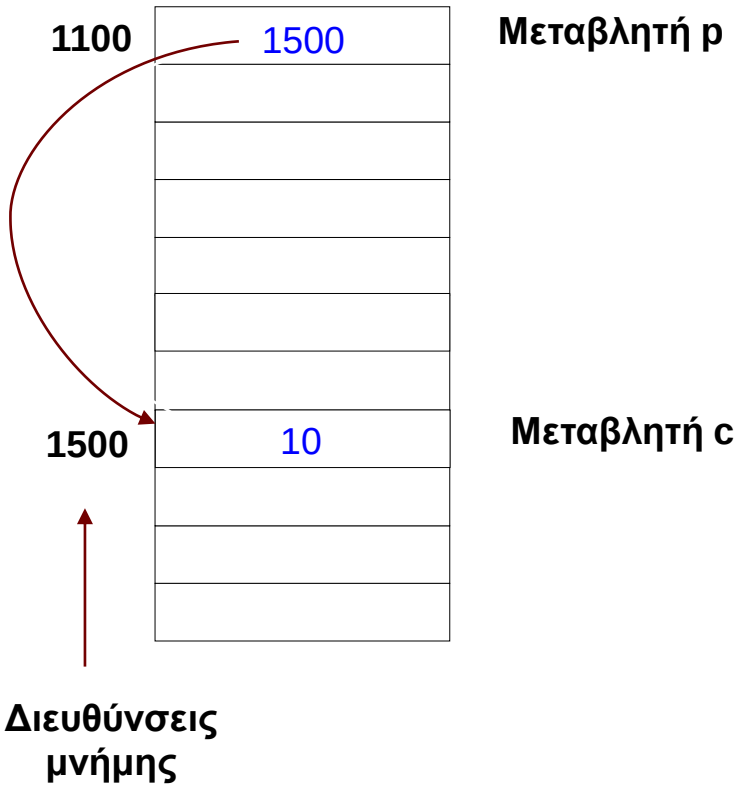
Τι μπορώ να κάνω με μία μεταβλητή (I)

- Η C είναι από τις ελάχιστες γλώσσες που *μας επιτρέπει να μάθουμε ποιο είναι το κελί* (διεύθυνση στη μνήμη) που φυλάει την τιμή μιας μεταβλητής `var`
  - Δίνεται από την έκφραση `&var`
  - Στην προηγούμενη διαφάνεια το &c είναι 1500, το &d είναι 1000

	Απλή μεταβλητή	
ΔΗΛΩΣΗ: Όνομα	`int var`	
ΣΕ ΕΚΦΡΑΣΗ: Τιμή	`var`	
ΣΕ ΕΚΦΡΑΣΗ: Διεύθυνση	`&var`	

Δείκτες - Pointers

- Δείκτης: τι είναι;
  - Μια μεταβλητή που περιέχει τη διεύθυνση μιας άλλης μεταβλητής
  - Δηλώνεται με ένα * πριν το όνομά του.

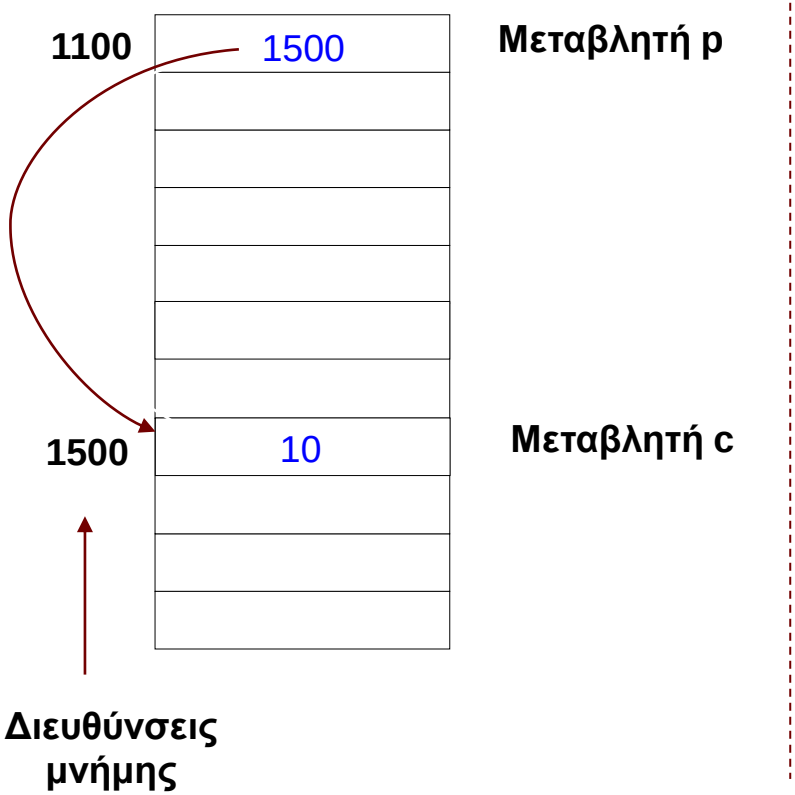



```
int c = 10, d;
int *p;  /* Δήλωση δείκτη */

p = &c;  /* Η δ/νση του c */
```

Δείκτες - Pointers

- Η διεύθυνση στην οποία βρίσκεται ο c → &c
- Τα περιεχόμενα (τιμή) του p → η διεύθυνση του c
- Τα περιεχόμενα του κελιού στο οποίο δείχνει ο p → *p




```
int c = 10, d;
int *p;  /* Δήλωση δείκτη */

p = &c;  /* Η δ/νση του c */
d = *p;  /* Έκφραση:
          * *p = ταξίδεψε όπου
          * δείχνει ο p και πάρε
          * πάρε το περιεχόμενο
          */
```

Μεταβλητές & μνήμη

```
#include <stdio.h>

int main() {
  int var1;
  char var2[10];

  printf("Address of var1 variable: %p\n", &var1 );
  printf("Address of var2 variable: %p\n", &var2 );
  return 0;
}
```

```
--------
Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6
```

Μεταβλητές & μνήμη

```
#include <stdio.h>

int main() {
  int var = 20; /* actual variable declaration */
  int * ip;     /* pointer variable declaration */

  ip = &var; /* store address of var in pointer variable*/
  printf("Address of var variable: %p\n", &var );
  printf("Address stored in ip variable: %p\n", ip );
  printf("Value of *ip variable: %d\n", *ip );
  return 0;
}
```

```
--------
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

Τι μπορώ να κάνω με μία μεταβλητή (II)

- Όπως όλες οι μεταβλητές, ο δείκτης έχει τιμή και μπορούμε να μάθουμε σε ποιο κελί μνήμης φυλάσσεται η τιμή του.
  - Επιπλέον όμως, μπορούμε να δούμε το περιεχόμενο του κελιού στο οποίο δείχνει.

	Απλή μεταβλητή	Δείκτης
ΔΗΛΩΣΗ: Όνομα	`int var`	`int *p`
ΣΕ ΕΚΦΡΑΣΗ: Τιμή	`var`	`p`
ΣΕ ΕΚΦΡΑΣΗ: Διεύθυνση	`&var`	`&p`
ΣΕ ΕΚΦΡΑΣΗ: Περιεχόμενο εκεί που δείχνει	`-`	`*p`

Αναλογία με αποθήκη προϊόντων [δεν ισχύει πλέον ☹]

- Γνωστό κατάστημα διαθέτει αποθήκη με πολλά ράφια όπου σε κάθε ράφι υπάρχει ένα προϊόν (έπιπλο). Τα προϊόντα είναι συσκευασμένα έτσι ώστε ΔΕΝ ΜΠΟΡΕΙΣ να καταλάβεις τι είναι το καθένα εαν πας στα ράφια της αποθήκης. Για να πάρεις το έπιπλο πρέπει να επισκεφτείς την έκθεση του καταστήματος και να σημειώσεις σε ειδικά χαρτάκια το ΡΑΦΙ της αποθήκης που έχει το προϊόν που σε ενδιαφέρει. Με το συμπληρωμένο χαρτάκι πας στο σωστό ράφι και το παίρνεις – αλλιώς δεν μπορείς να βρεις το προϊόν.
- Αναλογία:
  - **Μνήμη** = αποθήκη
  - **Μεταβλητή** (κελί στη μνήμη) = το ράφι (ο αριθμός του)
  - **Τιμή μεταβλητής** = προϊόν στο ράφι
  - **Δείκτης** = το χαρτάκι

Έννοιες & αναλογίες

Λειτουργία / έννοια	Αναλογία
p = &x;	Γράφω σε χαρτάκι το ράφι.
*p	Το προϊόν που υπάρχει στο ράφι («ταξιδεύω» εκεί και το βρίσκω).
q = &y;	Άλλο χαρτάκι που γράφει άλλο ράφι.
p = q;	Στο πρώτο χαρτάκι αλλάζω τι έγραψα και γράφω το ίδιο ράφι που γράφει το δεύτερο χαρτάκι.

temp = *a;
*a = *b;
*b = *temp;
	«Ταξιδεύω» στα ράφια που γράφουν τα χαρτάκια a και b και εναλλάσσω τα προϊόντα.

Δηλώσεις

- Δήλωση μεταβλητής: `<type> *<name>;`
- Παράδειγμα:

```
int * p, x, y;     /* Μόνο το p είναι δείκτης */
y = 3;
p = &y;
x = *p + 1;
```

- Προσοχή: η έκφραση a***b** είναι πολλαπλασιασμός!

Καταχώρηση τιμής σε δείκτες

- Σωστή καταχώρηση:

```
int c;
int * p;
p = NULL;    /* Χρειάζεται το stdio.h */
p = 0;             /* Ισοδύναμο με NULL */
p = &c;
```

- Λάθος:

```
p = 100;
p = c;
```

Δήλωση μαζί με αρχικοποίηση δεικτών

- Μπορείτε να δηλώσετε έναν δείκτη και ταυτόχρονα να τον αρχικοποιήσετε:

```
int c, * p = &c;     /* μην το κάνετε, μπερδεύει!! */
```

  - Είναι ισοδύναμο με:

```
int c, * p;
p = &c;
```

- **ΠΡΟΣΟΧΗ**:
  - Στις **δηλώσεις** το «*p» σημαίνει ότι το p είναι δείκτης. ΔΕΝ σημαίνει ότι πηγαίνει εκεί που δείχνει και παίρνει το περιεχόμενο!
  - Μόνο όταν το «*p» εμφανίζεται μέσα σε **πράξεις** έχει την έννοια του «πηγαίνω και παίρνω το περιεχόμενο»

- Για να μην υπάρχει σύγχυση, καλύτερα **να μην αρχικοποιείτε με αυτόν τον τρόπο τους δείκτες**. Το σωστότερο είναι να τους αρχικοποιείτε πάντα στην τιμή NULL και μετά να κάνετε ότι τροποποιήσεις θέλετε:

```
int c, * p = NULL;
p = &c;
```

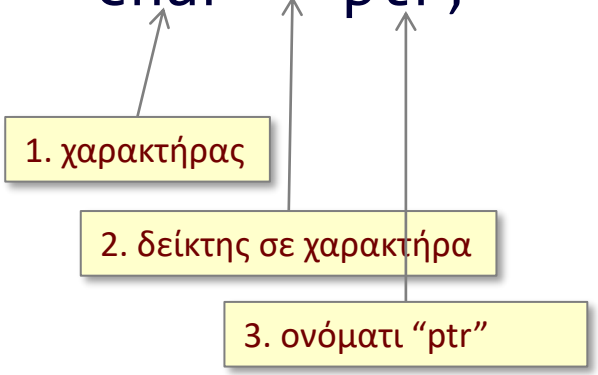
Δήλωση δείκτη

- Παράδειγμα:

```
char * ptr;   /* Ισοδύναμα: char* ptr, char *ptr */
```

- «Διαβάζοντας» τη δήλωση:

```
char * ptr;
```



Πρόκειται για μία **μεταβλητή** που ονομάζεται ptr.

Η μεταβλητή είναι **δείκτης.**

Η θέση στην κύρια μνήμη στην οποία θα δείχνει, αποθηκεύει έναν **char.**

Δείκτες και πέρασμα παραμέτρων

- Παραδείγματα:

```
void foo(char *s);
void bar(int *result);
```

- Γιατί να περάσω δείκτη;
  - Κλήση δια αναφοράς (call by reference)
  - Όταν θέλω, δηλαδή, η συνάρτηση να κάνει αλλαγές που να επηρεάζουν τα πραγματικά δεδομένα-ορίσματά της (δηλ. τις μεταβλητές της καλούσας συνάρτησης)
  - Επίσης, αν θέλω να περάσω ως όρισμα μία μεγάλη δομή αποφεύγοντας τη χρονοβόρα αντιγραφή στη στοίβα (αργότερα αυτά)

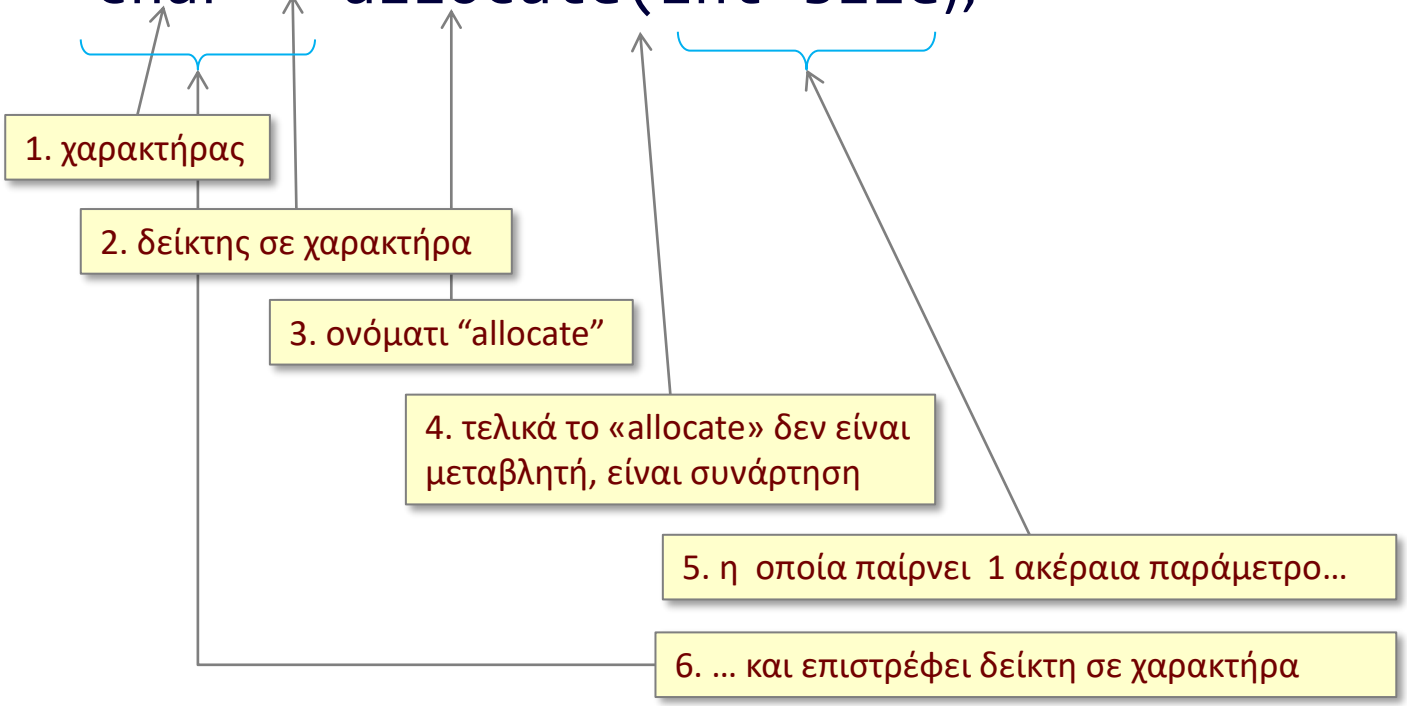
Συναρτήσεις που επιστρέφουν δείκτη

- Παράδειγμα:

```
char * allocate(int size);
```

- «Διαβάζοντας» τη δήλωση:

```
char * allocate(int size);
```



Ερωτήματα

- Υποθέστε τη δήλωση «`int x;`»
  Τι ακριβώς κάνουν τα παρακάτω αν βρεθούν σε μια έκφραση;
  - `&x;`
  - `*(&x);`
  - `&(*x);`

- Τι μεταβλητή χρειάζομαι για να βάλω μέσα τη διεύθυνση ενός δείκτη;

Κι άλλα ερωτήματα…

```
int i = 3, j = 5, k, * p=NULL, * q=NULL, * r=NULL;
p = &i; q = &j;
```

```
* * & p
```
ισοδύναμο: `*(*(&p))`
αποτέλεσμα: `3` (αφού είναι ισοδύναμο με `*(p)`)

```
3 * - * p / * q + 7
```
ισοδύναμο: `( (3 * ( - (*p))) / (*q) ) + 7`
αποτέλεσμα: `6`

```
3 * - * p /* q + 7
```
ισοδύναμο: `λάθος! Το /* ξεκινά σχόλιο!!`

```
* (r = & k) = *p * * q
```
ισοδύναμο: `(*(r = &k)) = ((*p) * (*q))`
αποτέλεσμα: `15`

Προτεραιότητες τελεστών C

Καλύτερα να **μην** θυμάστε τη σειρά!

Να καθορίζετε μόνοι σας τη σειρά των πράξεων βάζοντας στα σωστά σημεία **ΠΑΡΕΝΘΕΣΕΙΣ**.

Operator	Description	Associativity
`()`	Function call	Left to right
`[]`	Array element	
`->`	Structure member pointer reference	
`.`	Class, structure or union member reference	
`sizeof`	Storage size in bytes of object / type	
`++`	Postfix Increment	
`--`	Postfix Decrement	
`++`	Prefix Increment	Right to left
`--`	Prefix Decrement	
`-`	Unary minus	
`+`	Unary plus	
`!`	Logical negation	
`~`	One's complement	
`&`	Address of	
`*`	Indirection	
`(type)`	Type conversion (cast)	Right to left
`*`	Multiplication	Left to right
`/`	Division	
`%`	Modulus	
`+`	Addition	Left to right
`-`	Subtraction	
`<<`	Bitwise left shift	Left to right
`>>`	Bitwise right shift	
`<`	Scalar less than	Left to right
`<=`	Scalar less than or equal to	
`>`	Scalar greater than	
`>=`	Scalar greater than or equal to	
`==`	Scalar equal to	Left to right
`!=`	Scalar not equal to	
`&`	Bitwise AND	Left to right
`^`	Bitwise exclusive OR	Left to right
`|`	Bitwise inclusive OR	Left to right
`&&`	Logical AND	Left to right
`||`	Logical inclusive OR	Left to right
`?:`	Conditional expression	Right to left
`=`	Assignment	Right to left
`+= -= *=`	Assignment	
`/= %= &=`	Assignment	
`^= |=`	Assignment	
`<<= >>=`	Assignment	
`,`	Comma	Left to right

Πέρασμα παραμέτρων με αναφορά – swap

```
#include <stdio.h>
void swap (int x, int y);
int main() {
   int a, b;
   a = 6;
   b = 7;
   swap(a, b);
   printf("%d %d\n", a, b);
   return 0;
}
void swap (int x, int y) {
   int temp;
   temp = x;
   x = y;
   y = temp;
   return;
}
```

Πέρασμα παραμέτρων με αναφορά – swap

```
#include <stdio.h>
void swap (int x, int y);
int main() {
   int a, b;
   a = 6;
   b = 7;
   swap(a, b);
   printf("%d %d\n", a, b);
   return 0;
}
void swap (int x, int y) {
   int temp;
   temp = x;
   x = y;
   y = temp;
   return;
}
```

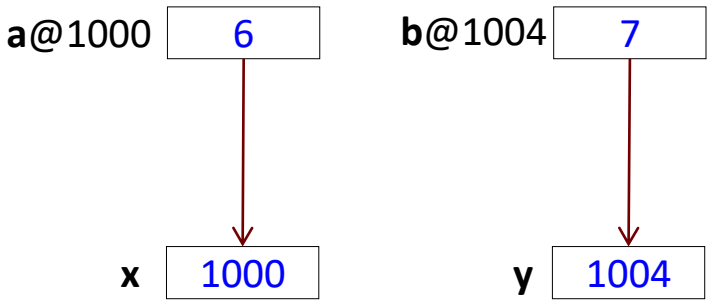
```
swap(&a, &b);
```

```
void swap (int *x1, int *x2) {

   temp = *x1;
   *x1 = *x2;
   *x2 = temp;
```

Swap με δείκτες

```
#include <stdio.h>
void swap (int *x, int *y);
int main() {
   int a, b;
   a = 6;
   b = 7;
   swap(&a, &b);
   printf("%d %d\n", a, b);
   return 0;
}
void swap (int *x, int *y) {
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
   return;
}
```



Τα x, y περιέχουν διευθύνσεις

Swap με δείκτες – εναλλακτικός κώδικας

```c
#include <stdio.h>
void swap (int *x, int *y);
int main() {
   int a, b;
   a = 6;
   b = 7;
   swap(&a, &b);
   printf("%d %d\n", a, b);
   return 0;
}
void swap (int *x, int *y) {
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
   return;
}
```

```c
#include <stdio.h>
void swap (int *x, int *y);
int main() {
   int a = 6, b = 7, *pa, *pb;
   pa = &a;
   pb = &b;
   swap(pa, pb);
   printf("%d %d\n", a, b);
   return 0;
}
void swap (int *x, int *y) {
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
   return;
}
```

Παράδειγμα

```c
#include <stdio.h>
int max(int x, int y);

int main() {
   int a, b, res;
   a = 5;
   b = 3;
   res = max(a, b);
   return 0;
}

int max(int x, int y) {
   int res;
   if (x > y) res = x;
   else res = y;
   return res;
}
```

```c
#include <stdio.h>
void max(int x, int y, int * res);

int main() {
   int a, b, res;
   a = 5;
   b = 3;
   max(a, b, &res);
   return 0;
}

void max(int x, int y, int *res) {
   if (x > y) *res = x;
   else *res = y;
}
```

Παράδειγμα

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

int main() {
   int a, b;
   init(&a, &b);
   return 0;
}

void init(int *x, int *y) {
  *x = 12;
  *y = 15;
}
```

Πέρασμα παραμέτρων με αναφορά - πίνακες

- Αν θυμάστε, το πέρασμα των πινάκων σε μία συνάρτηση γίνεται πάντα με αναφορά.
  - Αυτό που γίνεται είναι ότι περνιέται **ΜΟΝΟ ΕΝΑΣ ΔΕΙΚΤΗΣ ΣΤΟ ΠΡΩΤΟ ΣΤΟΙΧΕΙΟ ΤΟΥ ΠΙΝΑΚΑ** (μπορείτε να φανταστείτε γιατί;;)
  - Τα παρακάτω είναι ισοδύναμα:

```
void initzero(int x[100]) {   // Αγνοείται το μέγεθος!
  x[0] = 10;
}
void initzero(int x[]) {      // Άρα δεν χρειάζεται
  x[0] = 10;
}
void initzero(int *x) {       // Είναι απλός δείκτης
  *x = 10;
}
```

- *Μόνο σε παράμετρο συνάρτησης επιτρέπεται να μην δοθεί το πλήθος των γραμμών* ενός πίνακα διότι **ΑΓΝΟΕΙΤΑΙ** – ο πίνακας περνιέται ως ένας απλός δείκτης, *χωρίς πληροφορία για το πλήθος των στοιχείων του* πίνακα.

Παράδειγμα

```
#include <stdio.h>
void initarr(int x[10]);

int main() {
   int a[10];
   initarr(a);
   return 0;
}

void initarr(int x[10]) {
  int i;
  for (i = 0; i < 10; i++)
     x[i] = i;
}
```

```
#include <stdio.h>
void initarr(int x[], int n);

int main() {
   int a[10];
   initarr(a, 10);
   return 0;
}

void initarr(int x[], int n) {
  int i;
  for (i = 0; i < n; i++)
     x[i] = i;
}
```

Παρατήρηση

- Δεν έχει νόημα μια συνάρτηση να επιστρέφει δείκτη που δείχνει τοπικά, δηλαδή τη διεύθυνση μιας τοπικής μεταβλητής:

```
int * test1() {
 int i;
 return &i;
}
```

- Έχει διαφορά η παρακάτω περίπτωση;

```
int * test2() {
 static int i;
 return &i;
}
```

Παρένθεση - υπενθύμιση

- Το α και β παρακάτω είναι ίδια; Αν όχι υπάρχει κάποιο πρόβλημα;

(a)	(b)
`int x, * y = &x;`	`int x, * y;` `*y = &x;`

- Θυμηθείτε ότι άλλο σημαίνουν οι τελεστές της C μέσα σε μία ΔΗΛΩΣΗ και άλλο μέσα σε μία εντολή/πράξη. Το (b) λοιπόν είναι διαφορετικό (και λάθος). Το (a) θα ήταν ισοδύναμο με το (c):

(a)	(c)
`int x, * y = &x;`	`int x, * y;` `y = &x;`

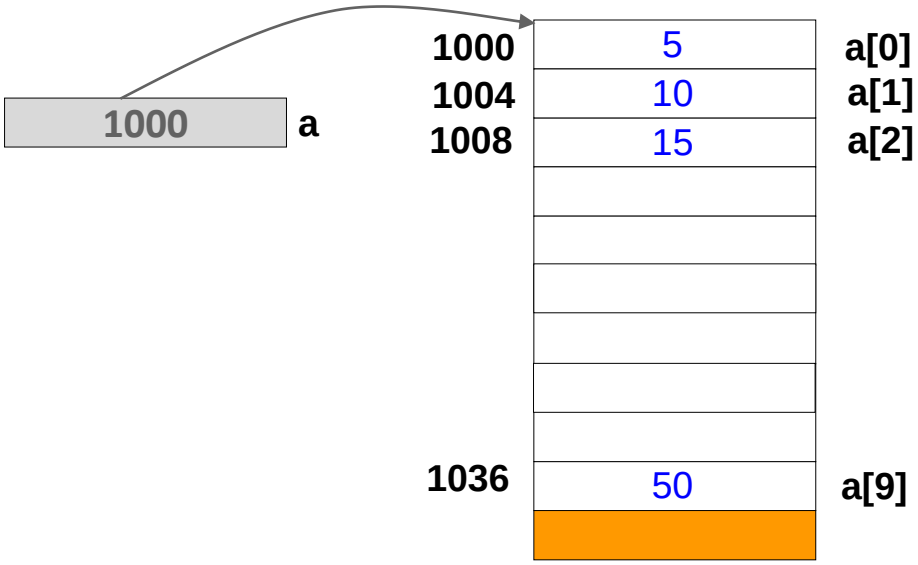
- *Καλό είναι να μην κάνουμε αρχικοποίηση ενός δείκτη κατά τη δήλωσή του (παρά μόνο με NULL).*

- Παράδειγμα:

```
int * p=NULL;
int a[10] = {5, 10, 15, 20, 25, 30, 35, 40, 45, 50};
p = &a[0];   /* διεύθυνση 1ου στοιχείου του πίνακα */
p = a;       /* ισοδύναμο με το παραπάνω */
```

p[i] <=> a[i]

- Τι είναι η μεταβλητή a? Είναι ένας **σταθερός** δείκτης που αντιστοιχεί σε μια διεύθυνση. *Ο σταθερός δείκτης* ***δεν μπορεί να δείξει κάπου αλλού***.



Ο τελεστής []

- Μέχρι τώρα γνωρίζαμε ότι ο τελεστής [] χρησιμοποιείται για επιλογή κάποιου στοιχείου ενός πίνακα, π.χ. το a[3] υποδηλώνει το 3ο στοιχείο του πίνακα.
- Όπως αυτή είναι η «μισή» αλήθεια:
  - Βασικά το a[3] σημαίνει **το στοιχείο που είναι 3 θέσεις μετά από εκεί που δείχνει (ξεκινά) ο δείκτης a**.
  - Επομένως, μπορεί να χρησιμοποιηθεί με γενικούς δείκτες και όχι μόνο με πίνακες!
  - Π.χ. αν p = &a[0], μπορώ να πω p[1] = 4;
  - Π.χ. αν p = &a[2], μπορώ να πω p[1] = 4;
- Η έκφραση **A[B]** είναι ισοδύναμη με ***(A+B)**
  - "Μυστικό": επομένως το a[2] είναι ισοδύναμο με το **2[a]** !!!

Παράδειγμα

```
#include <stdio.h>
main() {
  int src[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int * p=NULL, * q=NULL;
  int n = 3;

  p = & src[0];
  q = & src[2];
  printf("%d %d\n", *p, p[0]);
  printf("%d %d\n", *q, q[0]);

  p[n] += 11;
  printf("%d\n", p[n]);

  q[n] += 11;
  printf("%d\n", q[n]);
}
```

Εκτυπώσεις:

```
1 1
3 3

15

17
```

Αριθμητική δεικτών

(n ακέραιος)

- Μπορούμε να έχουμε εκφράσεις με δείκτες
  - p < q Ποιος δείχνει πιο μπροστά και ποιος πιο πίσω?
  - p + n Το κελί που βρίσκεται n **κελιά** πιο μετά
  - p – n Το κελί που βρίσκεται n **κελιά** πιο πριν
  - p – q Πόσο μακριά βρίσκονται (σε κελιά)?
- Άλλες δυνατές εκφράσεις
  - p = p + n
  - p += n
- Δεν δουλεύουν:
  - p + q
  - p*4

Αριθμητική δεικτών

```
char c, * pc=NULL;
int  i, * pi=NULL;

pc = &c; pi = &i;
printf("pc = %p, pc+1 = %p", pc, pc+1);
printf("pi = %p, pi+1 = %p", pi, pi+1);
```

Δίνει:

```
pc = 251af3c8, pc+1 = 251af3c9
pi = 251af3c4, pi+1 = 251af3c8
```

Γιατί;

Διότι το +1 δεν είναι +1 byte αλλά +1 τύπος (όσα bytes πιάνει ο τύπος του δείκτη)

```
printf("char: %d bytes", sizeof(char));
printf("int: %d bytes", sizeof(int));
```

Δίνει:

```
char: 1 bytes
int: 4 bytes
```

Παράδειγμα

```
#include <stdio.h>
main() {
  int src[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int * p=NULL, * q=NULL;
  int n = 3;

  p = & src[0];
  q = & src[2];
  printf("%d %d\n", *p, p[0]);
  printf("%d %d\n", *q, q[0]);

  *(p+n) += 11;
  printf("%d\n", p[n]);

  *(q+n) += 11;
  printf("%d\n", q[n]);
  ……
}
```

Εκτυπώσεις:

```
1 1
3 3

15

17
```

Συνεχίζεται...

Παράδειγμα

```
#include <stdio.h>
main() {
  int src[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int * p=NULL, * q=NULL;
  int n = 3;
  ……..                                  … από πριν
  if (q > p)
    printf("%p\n", q - p);
  else
    printf("%p\n", p - q);

  p++;
  q+=2;

  printf("%d %d\n", *p, p[0]);
  printf("%d %d\n", *q, q[0]);
}
```

Εκτυπώσεις:

```
2

2 2

5 5
```

Δείκτες και πίνακες, πάλι

- Υποθέτουμε ότι έχουν δηλωθεί: `int a[10], *p;`
- **Ισχύει**:

$$\texttt{\&a[i]} \Leftrightarrow \texttt{a+i}$$

$$\texttt{a[i]} \Leftrightarrow \texttt{*(a+i)}$$

- Αν έχω εκτελέσει την εντολή: `p = a`, τότε **ισχύει**:

$$\texttt{\&a[i]} \Leftrightarrow \texttt{a+i} \Leftrightarrow \texttt{\&p[i]} \Leftrightarrow \texttt{p+i}$$

$$\texttt{a[i]} \Leftrightarrow \texttt{*(a+i)} \Leftrightarrow \texttt{p[i]} \Leftrightarrow \texttt{*(p+i)}$$

- Ενώ ισχύει η έκφραση p++, **δεν ισχύουν** οι εκφράσεις: a++ και a = p,
  - παρά μόνο αν το a[] είναι παράμετρος σε μια συνάρτηση

Παράδειγμα, πάλι

```
#include <stdio.h>
void initarr(int x[], int n);

int main() {
   int a[10];
   initarr(a, 10);
   return 0;
}

void initarr(int x[], int n) {
  int i;
  for (i = 0; i < n; i++)
     x[i] = i;
}
```

```
#include <stdio.h>
void initarr(int *x, int n);

int main() {
   int a[10];
   initarr(a, 10);
   return 0;
}

void initarr(int *x, int n) {
  int i;
  for (i = 0; i < n; i++)
     x[i] = i;
}
```

Παρένθεση: ο τελεστής sizeof

- Το sizeof() (το οποίο δεν είναι συνάρτηση αλλά **ΤΕΛΕΣΤΗΣ** της C), επιστρέφει **το μέγεθος σε BYTES είτε ενός τύπου είτε μίας μεταβλητής:**

- Τι θα τυπωθεί παρακάτω (υποθέστε 32-μπιτο σύστημα);

```
char x, s[5], * p = NULL;
int y, a[10], * q = NULL;
p = &x; q = a;
printf("char: %d, int: %d",  sizeof(char), sizeof(int));
   char: 1, int: 4
printf("x: %d, y: %d",  sizeof(x), sizeof(y));
   x: 1, y: 4
printf("s: %d, a: %d",  sizeof(s), sizeof(a));
   s: 5, a: 40
printf("p: %d, q: %d",  sizeof(p), sizeof(q));
   p: 4, q: 4
printf("*p: %d, *q: %d",  sizeof(*p), sizeof(*q));
   p: 1, q: 4
```

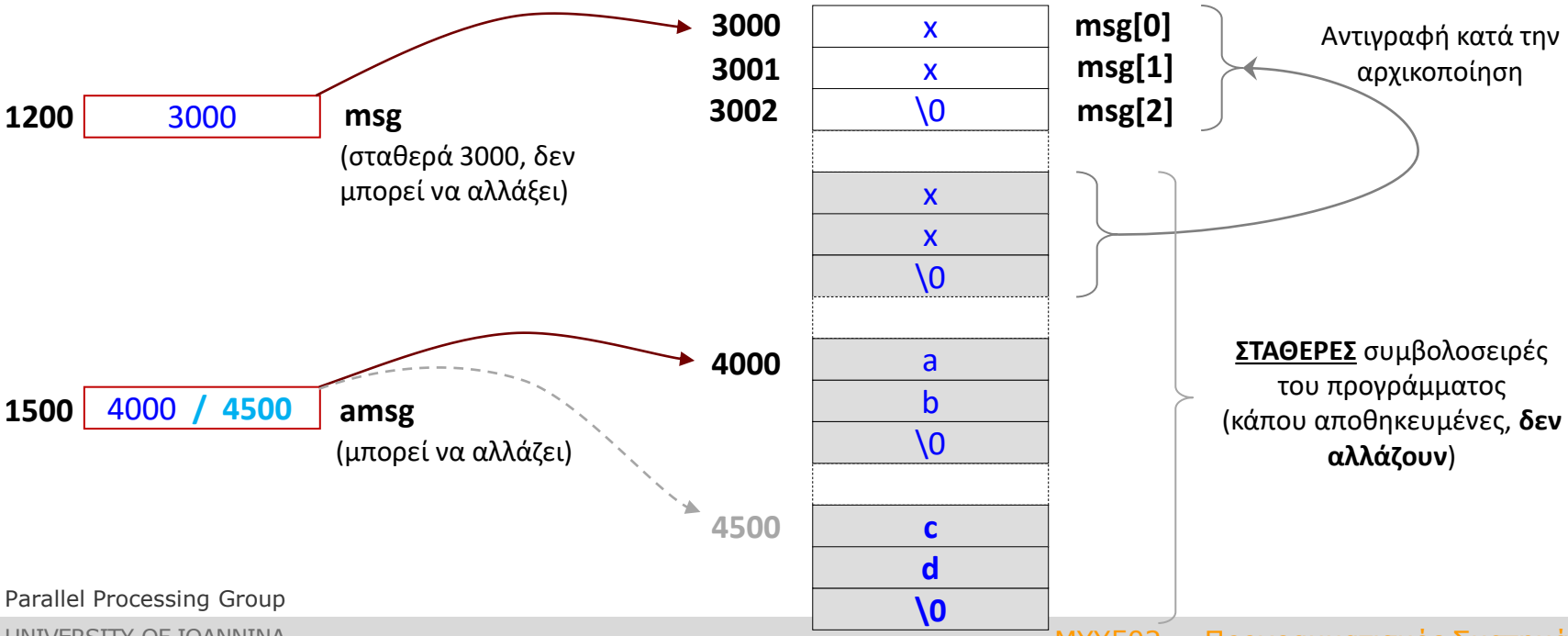
Τρικ με pointers

- Πώς μπορώ να δω αν το μηχάνημά μου αποθηκεύει με σειρά big endian ή little endian?
- *Homework ;-)*

Δείκτες και συμβολοσειρές

(ΥΠΕΝΘΥΜΙΣΗ: Κάθε string περιέχει και έναν παραπάνω χαρακτήρα, το '\0')

```
char msg[] = "xx";   /* Πίνακας 3 στοιχείων με αρχικοποίηση */
msg = "yyyy";        /* Δεν επιτρέπεται (σταθερός δείκτης)! */
char * amsg = "ab"; /* Δείκτης σε σταθερή συμβολοσειρά */
amsg = "cd";         /* OK – δείχνει σε άλλο χώρο! */
```



str.c

```
#include <stdio.h>
int main() {
    char buf[]  = "hello";
    char * ptr  = "geia sou";
    char * ptr1 = "file mou";

    buf[2] = '$';
    printf("%s\n", buf);

    ptr[2] = '%';           /* crash (why??) */

    ptr = ptr1;
    printf("%s\n", ptr);

    ptr[2] = '#';           /* crash (why??) */
    buf = ptr;              /* compile error (why??) */

    ptr = buf;
    ptr[2] = 'l';
    printf("%s\n", ptr);

    printf("buf occupies %d bytes in memory.\n", sizeof(buf));   /* ??? */
    printf("ptr occupies %d bytes in memory.\n", sizeof(ptr));   /* ??? */
    return 0;
}
```

Παραδείγματα

```
#include <stdio.h>
void strcpy1(char s[], char d[]);

int main() {
  char x[30] = "abcdef";
  char y[10];
  strcpy1(x, y);
  printf("%s", y);
  return 0;
}

void strcpy1(char s[], char d[]) {  /* Copy s to d */
   int i;
   i = 0;
   while (s[i] != '\0') {
        d[i] = s[i];
        i++;
   }
   d[i] = '\0';
}
```

Αντιγραφή – Εναλλακτικά

```
#include <stdio.h>
void strcpy2(char * s, char * d);

int main() {
  char x[30] = "abcdef";
  char y[10];
  strcpy2(x, y);
  printf("%s", y);
  return 0;
}

void strcpy2(char * s, char * d) {   /* Copy s to d */
   while (*s != '\0') {
        *d = *s;
        s++; d++;
   }
   *d = '\0';
}
```

Αντιγραφή – Εναλλακτικά

```
void strcpy3 (char s[], char d[]) {
  int i  = 0;
  while ((d[i] = s[i]) != '\0') {
          i++;
  }
}

void strcpy4 (char * s, char * d) {
  while ((*d = *s) != '\0') {
      s++; d++;
  }
}
```

Σύγκριση

```
int strcmp1(char s1[], char s2[]) {
   int i;
   for (i = 0; s1[i] == s2[i]; i++) {
       if (s1[i] == '\0')  { /* και τα δύο ίσα με \0 */
              return 0;
       }
   }
   return ( s1[i] – s2[i] );
}
```

- Η συνάρτηση επιστρέφει: 0 αν ΙΣΑ, < 0 αν s1 < s2, > 0 αν s1 > s2.
- Παράδειγμα: s1 → "abc", s2→"abcd"

Σύγκριση - Εναλλακτικά

```
int strcmp2(char * s1, char * s2) {
   for (; *s1 == *s2;  s1++, s2++) {
       if (*s1 == '\0') {
              return 0;
       }
   }
   return *s1 – *s2;
}

int main() {
   char A[N], B[N];
   …
   strcmp2(A, B);
   …
}
```

Μήκος

```
int strlen1(char s[]) {
   int i = 0;
   while (s[i] != '\0') i++;
   return i;
}

int strlen2(char * s) {
   char *p = s;
   while (*p != '\0') p++;       /* προχωράει 1 byte */
   return p-s ;
}
```

Πίνακες Δεικτών (Array of Pointers)

- Τι είναι;
  - Πίνακας που περιέχει δείκτες
- Πώς δηλώνεται;

```
<type> * <name> [size];
```

- Πώς αρχικοποιείται;
  - `int A[10];`
  - `int B[20];`
  - `int * C[2] = { A, B };`

- Αντί για πίνακα 2 διαστάσεων χρησιμοποιώ πίνακα δεικτών
  - Οικονομία μνήμης όταν η 2η διάσταση μεταβάλλεται
  - Όταν η 2η διάσταση είναι άγνωστη

Παραδείγματα

- Πίνακας με strings διαφορετικών μεγεθών

```
char * months[12] = {"January", "February", "March", ..};
```

Παράδειγμα

```
char * get_month(int i) {
    static char * months[12] = {"Jan", "Feb", …};
    return ( (i<12 && i >=0) ? months[i] : NULL );
}
```

- Τι είναι η τιμή NULL που μπορεί να επιστρέψει η συνάρτηση?
- Τι θα συμβεί αν βγάλω το static;

Επανάληψη – υπενθύμιση: τι δουλεύει και τι όχι;

```
#include <stdio.h>
int main() {
    char buf[] = "hello";                  // Αντιγραφή από σταθερή συμβολοσειρά
    char * ptr = "hello";                  // Δείχνει στον 1ο χαρακτήρα της σταθερ. συμβολ.
    void test1(char s[]);
    void test2(char *s);

    buf = ptr;                             // Δεν επιτρέπεται να αλλάξει ο «δείκτης» buf
    ptr = buf;                             // OK, ο δείκτης ptr θα δείχνει όπου και ο buf
    buf = "dolly";                         // Δεν επιτρέπεται να αλλάξει ο δείκτης buf
    ptr = "dolly";                         // OK, θα δείχνει όπου είναι η αρχή του “dolly”
    *buf = 'D';                            // OK, ισοδύναμο με buf[0] = 'D';
    *ptr = 'D';                            // Δεν επιτρέπεται αλλαγή στη σταθερή συμβολοσ.
    printf("%d", sizeof(buf));             // 6 (5 χαρακτήρες + το \0)
    printf("%d", sizeof(ptr));             // 4 (σε 32-μπιτο, 4 bytes για αποθήκευση δείκτη)
    test1(buf);
    test1(ptr);
    test2(buf);
    test2(ptr);
    return 0;
}

void test1(char s[]) {                     // Πίνακας-παράμετρος περνιέται ως απλός δείκτης
    printf("%d, %c", sizeof(s), *(s+1)); // 4 (σε 32-μπιτο, 4 bytes για αποθήκευση δείκτη)
}

void test2(char * s) {
    printf("%d, %c", sizeof(s), s[1]);   // 4 (σε 32-μπιτο, 4 bytes για αποθήκευση δείκτη)
}
```

Είναι όλα OK?

```c
#include <stdio.h>
void max(int x, int y, int * res);

int main() {
   int a, b, res;
   a = 5;
   b = 3;
   max(a, b, &res);
   return 0;
}

void max(int x, int y, int *res) {
   if (x > y) *res = x;
   else *res = y;
}
```

```c
#include <stdio.h>
void max(int x, int y, int * res);

int main() {
   int a, b, *res;
   a = 5;
   b = 3;
   max(a, b, res);
   return 0;
}

void max(int x, int y, int *res) {
   if (x > y) *res = x;
   else *res = y;
}
```