

Θεματική ενότητα 9: **Δείκτες**

Εισαγωγή

Σε κάθε μεταβλητή αποδίδεται μία θέση στην κύρια μνήμη του υπολογιστή, η οποία χαρακτηρίζεται από τη διεύθυνσή της. Στη γλώσσα μηχανής μπορεί να γίνει άμεση χρήση αυτής της διεύθυνσης, για να αποθηκευτούν ή να ανακληθούν δεδομένα. Αντίθετα, στις γλώσσες προγραμματισμού υψηλού επιπέδου οι διευθύνσεις δεν είναι άμεσα ορατές από τον προγραμματιστή, καθώς καλύπτονται από τον μανδύα των συμβολικών ονομάτων, τα οποία το σύστημα αντιστοιχεί στις πραγματικές διευθύνσεις.

Η γλώσσα C υποστηρίζει την άμεση διαχείριση των περιεχομένων της μνήμης εισάγοντας την έννοια του δείκτη (pointer). **Ο δείκτης αποτελεί μία ιδιάζουσα μορφή μεταβλητής, η οποία έχει ως περιεχόμενο όχι ένα πραγματικό δεδομένο αλλά μία διεύθυνση μνήμης.** Οι δείκτες είναι ένα ισχυρό προγραμματιστικό εργαλείο και εφαρμόζονται:

- στη δυναμική εκχώρηση μνήμης,
- στη διαχείριση σύνθετων δομών δεδομένων,
- στην αλλαγή τιμών που έχουν εκχωρηθεί ως ορίσματα σε συναρτήσεις,
- για τον αποτελεσματικότερο χειρισμό πινάκων.

Ωστόσο, καθώς η χρήση των δεικτών οδηγεί σε επεμβάσεις στη μνήμη και πολλές φορές σε προγραμματιστικές ακροβασίες, συχνά αποτελεί αιτία δύσκολων στον εντοπισμό σφαλμάτων, γι' αυτό και πρέπει να γίνεται με ιδιαίτερη προσοχή.

Δείκτες (pointers)

- Πίνακες ως παράμετροι συναρτήσεων:

Περνά τη διεύθυνση του πίνακα

```
main () {  
    int nums[5]={1,2,3,4,5};  
    clear(nums, 5);  
}
```


```
void clear(int ar[ ], int size) {  
    int i;  
    for (i=0; i < size; i++)  
        ar[i]=0;  
}
```

Αλλάζει τα στοιχεία του nums

Δείκτες

- Μπορούμε να αλλάξουμε τις τιμές που είναι αποθηκευμένες στον πίνακα **nums** μέσα από τη συνάρτηση **clear()**.
- Αυτό συμβαίνει γιατί το όνομα **nums** αναφέρεται στη **διεύθυνση μνήμης**, από την οποία ξεκινά ο πίνακας.

```
main () {  
    int x=3; y=5;  
    swap(x,y);  
    printf("x=%d, y=%d", x, y);  
}
```



```
void swap (int a, int b) {  
    int temp = a;  
    a = b;  
    b = temp;  
}
```

Τυπώνει **x=3, y=5**. Δε γίνεται η ανταλλαγή (swapping)!

Δείκτες

- Το πρόβλημα συμβαίνει επειδή απλώς αντιγράφονται οι τιμές των x και y στα ορίσματα a και b .
- Άρα, τι θα γινόταν εάν μπορούσαμε με κάποιον τρόπο να ανταλλάξουμε τις διευθύνσεις των x και y ;
 - Τότε, θα μπορούσαμε να αλλάξουμε τα x και y μέσα στη συνάρτηση `swap()`, όπως συνέβη με τον πίνακα.

Δήλωση δείκτη

- Δείκτης: μία μεταβλητή που κρατά **μία διεύθυνση**.

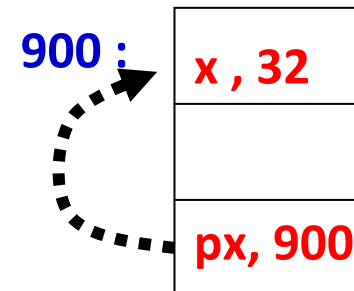
Διεύθυν. Περιεχόμ.

900 :	x , 32	← Κανονική μεταβλητή.
904 :		Όνομα: x, Τιμή: 32, Τύπος: int
908 :	px, 900	← Μεταβλητή δείκτη.

Όνομα: px, Τιμή: 900, Τύπος: δείκτης σε int

Λέμε ότι ο px δείχνει στην x

Φανταστείτε ένα τόξο από τη μεταβλητή δείκτη στην κανονική μεταβλητή, το οποίο δείχνει πού «δείχνει» ο δείκτης.



Δήλωση δείκτη

- Ο δείκτης πρέπει να δηλωθεί:

```
base_type * pointer_name ;
```

ο **τύπος** της μεταβλητής αποθηκεύεται στη θέση που δείχνει ο δείκτης

το **όνομα** της μεταβλητής δείκτη. **Καλή προγραμματιστική πρακτική:** το πρώτο γράμμα του ονόματος να είναι πάντοτε **p**

προσδιορίζει ότι δηλώνεται μία μεταβλητή **δείκτη**

Δήλωση δείκτη

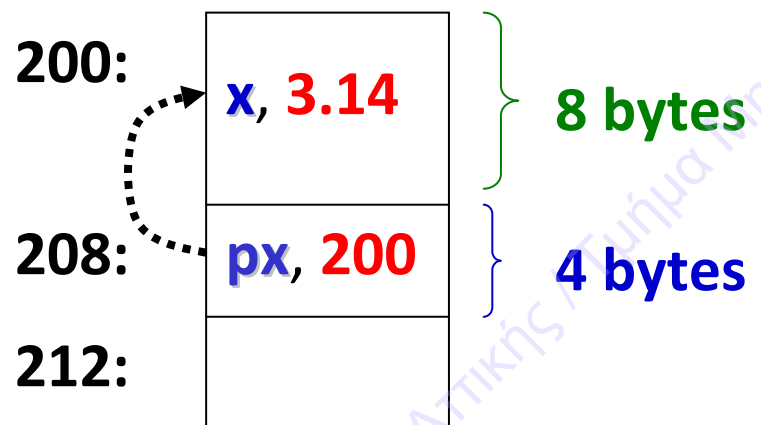
Γιατί πρέπει να δηλωθεί ο τύπος της κανονικής μεταβλητής;

- Γιατί όταν δηλώνεται μία (κανονική) μεταβλητή, δεσμεύεται συγκεκριμένη μνήμη, π.χ. 8 bytes για double, 4 bytes για int.
- Ο δείκτης αναφέρεται σε μία διεύθυνση, στην οποία αποθηκεύεται η τιμή μίας κανονικής μεταβλητής.
- Ο δείκτης χρησιμοποιείται για να γίνεται έμμεση αναφορά σ' αυτήν την τιμή. Έτσι, πρέπει να γνωρίζουμε πόση ακριβώς μνήμη καταλαμβάνει αυτή η τιμή.

Δήλωση δείκτη

Πόση μνήμη καταλαμβάνει η ίδια η μεταβλητή δείκτη;

- Η διεύθυνση είναι ένας ακέραιος, έτσι ο δείκτης καταλαμβάνει 4 bytes, ανεξάρτητα από τον τύπο της κανονικής μεταβλητής που δείχνει.



Η **x** είναι μεταβλητή τύπου double

Ο **px** είναι δείκτης στην **x**

Αναλογία με την πραγματικότητα : μία οκταμελής οικογένεια δε χρειάζεται μεγαλύτερη ταχυδρομική διεύθυνση από μία τετραμελή (ίσως μεγαλύτερο γραμματοκιβώτιο!)

Δήλωση δείκτη

Γιατί είναι απαραίτητος ο αστερίσκος;

*Διότι προσδιορίζει ότι δηλώνεται μία μεταβλητή με δείκτη
κι όχι μία κανονική μεταβλητή.*

ΠΡΟΣΟΧΗ : Αν και η μεταβλητή με δείκτη περιέχει μία διεύθυνση (ακέραιος αριθμός), **ΔΕΝ ΕΙΝΑΙ ΙΔΙΑ** με μία κανονική ακέραια μεταβλητή. Ο μεταγλωττιστής γνωρίζει ότι **η τιμή της μεταβλητής με δείκτη είναι μία συγκεκριμένη διεύθυνση μνήμης**, σε αντιδιαστολή με την «κανονική» ακέραια τιμή.

Δήλωση δείκτη

Ο αστερίσκος συνδέεται με το όνομα κι όχι με τον τύπο:

```
int *pcount; // δείκτης σε ακεραίους, με ονομασία pcount
```

```
int *pcount, *pnum; /* δείκτες σε ακεραίους, με ονομασίες pcount και pnum */
```

```
int *pcount, number; /* ένας δείκτης σε ακέραιο, με ονομασία pcount και ένας ακέραιος με ονομασία number */
```

Δήλωση δείκτη

Πώς επιλέγεται το όνομα ενός δείκτη;

- Οι ίδιες συμβάσεις που ισχύουν στις κανονικές μεταβλητές.
- Ωστόσο, συνήθως ο αρχικός χαρακτήρας του ονόματος δείκτη είναι το **p**, έτσι ώστε το πρόγραμμα να καθίσταται περισσότερο ευανάγνωστο, καθώς με τον πρώτο χαρακτήρα φαίνεται εάν μία μεταβλητή είναι δείκτης ή όχι. Εναλλακτικά, μπορούμε να προσθέτουμε την κατάληξη **_ptr**.

Παράδειγμα:

- **int *pcount, *count_ptr;** /* δείκτες σε ακεραίους */
- **char *pword, *word_ptr;** /* δείκτες σε χαρακτήρες */

Αρχικοποίηση δεικτών

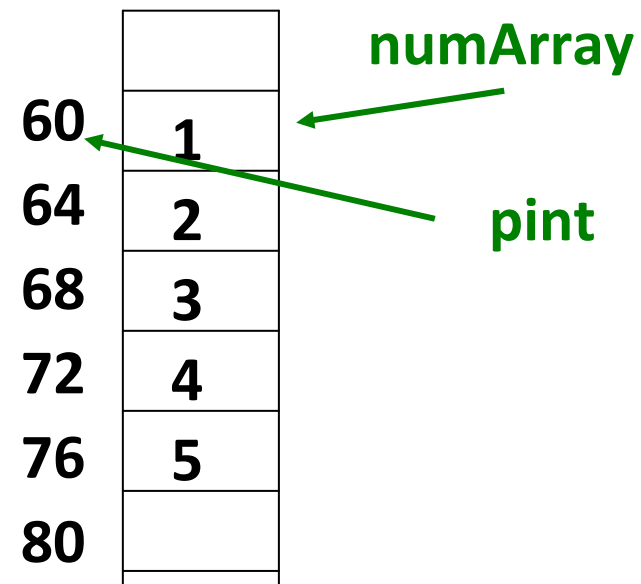
1) Χρησιμοποιώντας πίνακα

(Υπενθύμιση: το όνομα ενός πίνακα είναι μία διεύθυνση.)

```
int numArray[5] = {1,2,3,4,5};
```

```
int *pint;
```

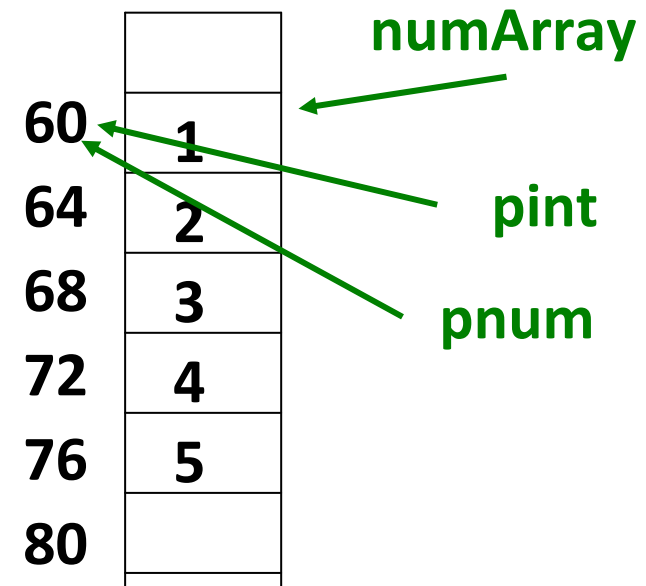
```
pint = numArray;
```



Αρχικοποίηση δεικτών

2) Χρησιμοποιώντας άλλους δείκτες ίδιου τύπου

```
int numArray[5] = {1,2,3,4,5};  
int *pint, *pnum;  
pint = numArray;  
pnum = pint;
```

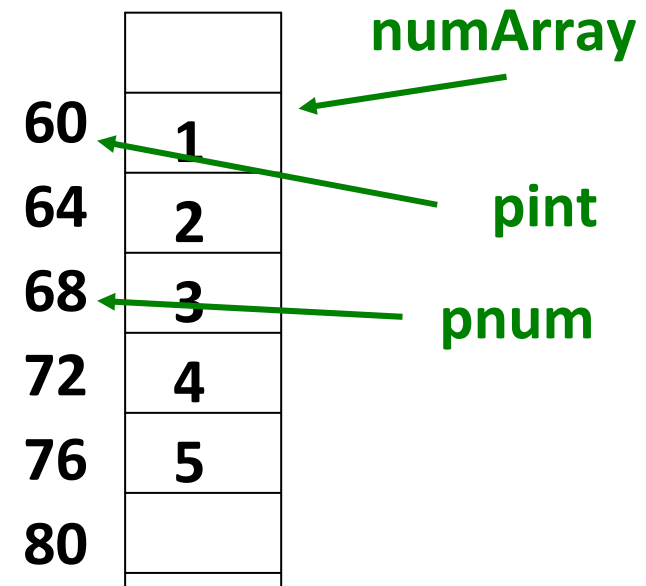


Αρχικοποίηση δεικτών

3) Χρησιμοποιώντας αριθμητική δεικτών

```
int numArray[5] = {1,2,3,4,5};  
int *pint, *pnum;  
pint = numArray;  
pnum = pint+2;
```

Πήγαινε δύο θέσεις ακεραίων
πιο κάτω



Αρχικοποίηση δεικτών

4) Χρησιμοποιώντας τον τελεστή διεύθυνσης & (address-of operator)

```
int *pnum;
```

```
int count;
```

```
pnum = &count;
```



Η γραμμή αυτή αναφέρει: ο **pnum** να λάβει ως τιμή τη διεύθυνση της μεταβλητής **count**, δηλαδή ο δείκτης **pnum** να «δείχνει» στη μεταβλητή **count**.

Ακολουθως φαίνεται πώς μπορούμε να προσπελάσουμε την τιμή μίας μεταβλητής με χρήση δείκτη.

```
int *pcount, num;
```

```
num = 10;
```

```
pcount = &num;
```

```
*pcount = 20;
```

900 :

pcount, *junk*

904 :

num, *junk*

908 :

...

```
int *pcount, num;
```

```
num = 10;
```

```
pcount = &num;
```

```
*pcount = 20;
```

```
int *pcount, num;
```

```
num = 10;
```

```
pcount = &num;
```

```
*pcount = 20;
```

900 : pcount, *junk*

904 : num, 10

908 :

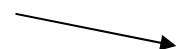
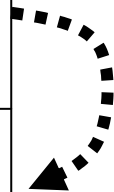
...

900 : pcount, 904

904 : num, 10

908 :

...



```
int *pcount, num;
```

```
num = 10;
```

```
pcount = &num;
```

```
*pcount = 20;
```

900 : pcount, 904

904 : num, 20

908 :

...

Ο αστερίσκος υποδηλώνει ότι πρέπει να ακολουθηθεί το βέλος για να προσπελασθούν τα δεδομένα της θέσης, στην οποία δείχνει.

***pcount = 20;**

- Ο αστερίσκος ονομάζεται **τελεστής περιεχομένου** (dereferencing operator).
- Διαβάζεται «στη διεύθυνση».
- Χρησιμοποιείται για να προσπελαύνει τα περιεχόμενα της θέσης μνήμης στην οποία δείχνει ο δείκτης.
- Δε θα πρέπει να συγχέεται με τον αστερίσκο της δήλωσης δείκτη.

Εφαρμογή δεικτών

Στο παράδειγμα της συνάρτησης `swap()` κατέστη φανερό ότι έπρεπε να γίνουν διορθώσεις:

- Μπορούμε να χρησιμοποιήσουμε δείκτες για να μεταβάλλουμε **έμμεσα** τις τιμές των μεταβλητών της `main()` μέσα από τη συνάρτηση `swap()`.
- Γνωρίζουμε πλέον:
 - Πώς ορίζονται οι δείκτες.
 - Πώς τους αρχικοποιούμε με τη διεύθυνση μίας μεταβλητής.
 - Πώς προσπελάζουμε την τιμή της μεταβλητής.

Εφαρμογή δεικτών

Τμήμα της *main()*

```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

Τα ορίσματα της συνάρτησης είναι τώρα δείκτες σε ακέραιες μεταβλητές.

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Θα χρησιμοποιήσουμε μαύρο χρώμα για τις τοπικές μεταβλητές της *main()* και μπλε για τις τοπικές μεταβλητές της *swap()*.



Τμήμα της main()

```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Απεικόνιση της μνήμης

address var name, value

900: x, 10

904: y, 25

908:

912:

916:

920:

924:

Τμήμα της main ()

```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Απεικόνιση της μνήμης

address var name, value

900: x, 10

904: y, 25

908: px, *junk*

912: py, *junk*

916:

920:

924:

Τμήμα της main ()

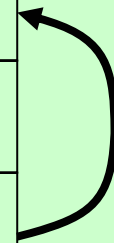
```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Απεικόνιση της μνήμης

address var name, value

900:	x, 10
904:	y, 25
908:	px, 900
912:	py, junk
916:	
920:	
924:	



Τμήμα της main ()

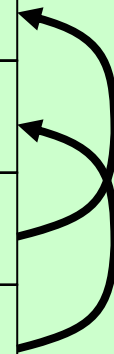
```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Απεικόνιση της μνήμης

address var name, value

900:	x, 10
904:	y, 25
908:	px, 900
912:	py, 904
916:	
920:	
924:	



Τμήμα της main ()

```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

Αντίγραφο της τιμής της
py

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Απεικόνιση της μνήμης

address var name, value

900:	x, 10
904:	y, 25
908:	px, 900
912:	py, 904
916:	pa, 900
920:	pb, 904
924:	

Τμήμα της main ()

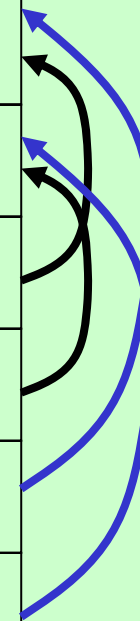
```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Απεικόνιση της μνήμης

address var name, value

900:	x, 10
904:	y, 25
908:	px, 900
912:	py, 904
916:	pa, 900
920:	pb, 904
924:	temp, junk



Τμήμα της main ()

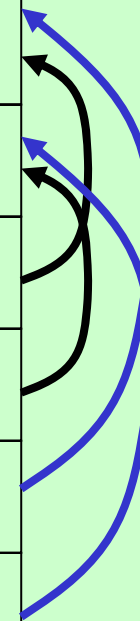
```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Απεικόνιση της μνήμης

address var name, value

900:	x, 10
904:	y, 25
908:	px, 900
912:	py, 904
916:	pa, 900
920:	pb, 904
924:	temp, 10



Τμήμα της main ()

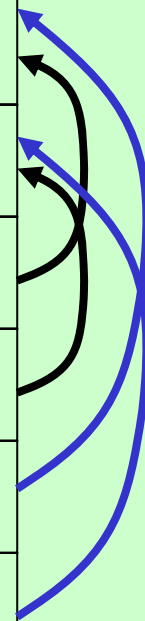
```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Απεικόνιση της μνήμης

address var name, value

900:	x, 25
904:	y, 25
908:	px, 900
912:	py, 904
916:	pa, 900
920:	pb, 904
924:	temp, 10



Τμήμα της main ()

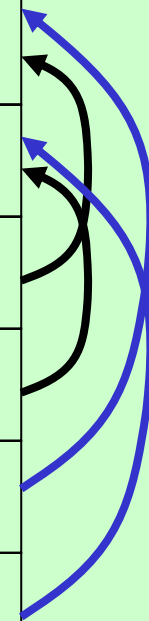
```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Απεικόνιση της μνήμης

address var name, value

900:	x, 25
904:	y, 10
908:	px, 900
912:	py, 904
916:	pa, 900
920:	pb, 904
924:	temp, 10



Τμήμα της main ()

```
{  
    int x=10, y=25;  
    int *px, *py;  
    px = &x;  
    py = &y;  
    swap(px, py);  
}
```

```
void swap (int *pa, int *pb) {  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Απεικόνιση της μνήμης

address var name, value

900:	x, 25
904:	y, 10
908:	px, 900
912:	py, 904
916:	
920:	
924:	



Επεξηγήσεις:

- Αν και η συνάρτηση `swap()` δεν επιστρέφει τίποτε άμεσα στη `main()`, έχει μία **παρενέργεια (side effect)**.
- Όταν καλείται η `swap()`, τα ορίσματά της είναι οι δείκτες `px` και `py`, οι οποίες σχετίζονται με τις διευθύνσεις των `x` και `y`, αντίστοιχα.
- **Παρατήρηση:** δε χρειάζεται να δηλώσουμε τους δείκτες `px` και `py`. Το μόνο που απαιτείται είναι να περασθούν οι διευθύνσεις των `x` και `y` στη `swap()`, όπως φαίνεται ακολούθως:

```
int main ()  
{  
    int x=10, y=25;  
    swap(&x, &y);  
    return 0;  
}
```

```
void swap (int *pa, int *pb)  
{  
    int temp;  
    temp = *pa;  
    *pa = *pb;  
    *pb = temp;  
}
```

Λειτουργεί με τον ίδιο τρόπο που λειτουργούσε το πρόγραμμα με τους **px** και **py**.

Δείκτες και συναρτήσεις

- Έως τώρα περνούσαμε τους δείκτες ως ορίσματα σε συναρτήσεις. Μπορούμε όμως να επιστρέφουμε δείκτες;
- Όταν επιστρέφεται ένας δείκτης, θα πρέπει να δείχνει σε δεδομένα της **καλούσας** συνάρτησης.
- **Δεν πρέπει ποτέ να επιστρέφεται δείκτης που δείχνει σε τοπική μεταβλητή** της **καλούμενης** συνάρτησης, γιατί όταν τερματισθεί η συνάρτηση οι τοπικές μεταβλητές εξαφανίζονται.
- Μπορούμε να χρησιμοποιήσουμε δείκτη για να αλλάξουμε το περιεχόμενο της θέσης στην οποία δείχνει, αλλά **δεν πρέπει να αλλάξουμε τον ίδιο τον δείκτη** μέσα στην καλούμενη συνάρτηση.

```
int main () {  
    int *pscore, num;  
    num = 32;  
    pscore = incr(num);  
    return 0;  
}
```

```
int *incr(int x) {  
    x = x+10;  
    return &x;  
}
```



ΛΑΘΟΣ! Όταν τελειώνει η *incr*, η *x* εξαφανίζεται και η τιμή της χάνεται.

Ωστόσο, πίσω στη συνάρτηση ο *pscore* θα δείχνει στη διεύθυνση που επέστρεψε από τη συνάρτηση αλλά θα υπάρχει «σκουπίδι» (junk) σ' αυτή τη διεύθυνση, εφόσον το *x* έχει εξαφανισθεί.

Σωστή χρήση της επιστρεφόμενης τιμής διεύθυνσης



```
/* pointer_func.c */  
#include <stdio.h>  
  
int *incr(int *pkitsos);  
int main()  
{  
    int *pscore, *pm, num;  
    num=32;  
    pscore=&num;  
    printf( "addr(num)=%d addr(pscore)=%d addr(pm)=%d  
    pscore=%d\n\n*pscore=%d\n\n",&num,&pscore,&pm,pscore,*pscore );  
    pm=incr(pscore);  
    printf( "pm=%d num=%d\n\n\n",pm,num );  
    return 0;  
}
```



```
int *incr(int *pk) {  
    int x;  
    x=*pk;  
    printf( "Prior: addr(pk)=%d  pk=%d  addr(x)=%d  
x=%d\n\n",&pk,pk,&x,x);  
    x=x+10;  
    *pk=x;  
    printf( "*pk=%d\n\n",*pk);  
    return(pk);  
}
```

```
addr(num)=6487564  addr(pscore)=6487576  addr(pm)=6487568  pscore=6487564
```

```
*pscore=32
```

```
Prior: addr(pk)=6487504  pk=6487564  addr(x)=6487484  x=32
```

```
*pk=42
```

```
pm=6487564  num=42
```

pointer_return_value.c

Η διαδικασία *callback* περιγράφεται στην υποενότητα 6.8 του βιβλίου, αλλά εκφεύγει του αντικειμένου του μαθήματος.

(Υπενθύμιση) Συνάρτηση και Πίνακες

Εάν η **πραγματική παράμετρος** είναι όνομα πίνακα (πχ. **key**), στέλνει τη διεύθυνση του πρώτου byte του πίνακα.

Η **παράμετρος στη δήλωση της συνάρτησης** είναι ένα όνομα **τοπικού** πίνακα (π.χ. **x**). Κρατά ένα αντίγραφο της ίδιας διεύθυνσης αλλά χρησιμοποιεί διαφορετικό όνομα.

Πραγματική παράμετρος

```
main ()
{
    float key[10];
    setKeyA(key,4);
}
/*key has 10 varied #s */
```

Παράμετρος συνάρτησης

```
setKeyA(float x[], int s)
{ /* 10 seeded random #s */
    int i;
    srand(s); /* seed it */
    for(i=0; i<10; i++)
        x[i] = rand();
}
```


Κλήση κατ' αξία

- Το όρισμα **4** αντιγράφει μία **ΤΙΜΗ** στη συνάρτηση. Η διαδικασία αυτή ονομάζεται **κλήση κατ' αξία (call by value)**.

Πραγματική παράμετρος  Παράμετρος συνάρτησης

```
Τμήμα της main ()  
{  
  float key[10];  
  setKeyA(key,4);  
}  
/*key has 10 varied #s */
```

```
setKeyA(float x[], int s)  
{ /* 10 seeded random #s */  
  int i;  
  srand(s); /* seed it */  
  for(i=0; i<10; i++)  
    x[i] = rand();  
}
```

Κλήση κατ' αξία

Το όρισμα **key** αντιγράφει τη **ΔΙΕΥΘΥΝΣΗ** σε μία συνάρτηση. Η συνάρτηση χρησιμοποιεί αυτή τη διεύθυνση για να βρει το προς χρήση δεδομένο.

Πραγματική παράμετρος

```
Τμήμα της main ()  
{  
  float key[10];  
  setKeyA(key,4);  
}  
/*key has 10 varied #s */
```

Παράμετρος συνάρτησης

```
setKeyA(float x[], int s)  
{ /* 10 seeded random #s */  
  int i;  
  srand(s); /* seed it */  
  for(i=0; i<10; i++)  
    x[i] = rand();  
}
```

Κλήση κατ' αναφορά

Όταν τα ορίσματα είναι πίνακες, περνιούνται στις συναρτήσεις **‘κατ’ αναφορά’ (call by reference)** — και μπορεί να είναι ΤΟΣΟ είσοδοι ΟΣΟ και έξοδοι.

(τόσο η `main()` όσο και η `setKeyA()` μπορούν να θέσουν τιμές σε στοιχεία πινάκων).

Πραγματική παράμετρος

```
Τμήμα της main ()  
{  
  float key[10];  
  setKeyA(key,4);  
}  
/*key has 10 varied #s */
```

Παράμετρος συνάρτησης

```
setKeyA(float x[], int s)  
{ /* 10 seeded random #s */  
  int i;  
  srand(s); /* seed it */  
  for(i=0; i<10; i++)  
    x[i] = rand();  
}
```

Κλήση κατ' αναφορά

- Οι δείκτες επιτρέπουν να περνάμε **ΟΠΟΙΟΔΗΠΟΤΕ δεδομένο κατ' αναφορά**
- Παράδειγμα: αλλάζουμε την **setKeyA** έτσι ώστε να δέχεται ως ορίσματα μόνο δείκτες και διευρύνονται οι δυνατότητες. **(αλλάζουμε το όνομα σε setKeyP)**

Πραγματική παράμετρος

Τμήμα της main ()

```
{ /*double-width key */  
int seed0 = 4; seed1 = 89;  
float key2[20];  
setKeyP(&key2[0], &seed0);  
setKeyP(&key2[10], &seed1);  
}
```

Παράμετρος συνάρτησης

```
setKeyP(float *pK, int *pS)  
{ /* 10 seeded random #s */  
int i;  
srand(*pS); /* seed... */  
for(i=0; i<10; i++)  
    pK[i] = rand();  
*pS=0; /* clear it */  
}
```

Κλήση κατ' αναφορά

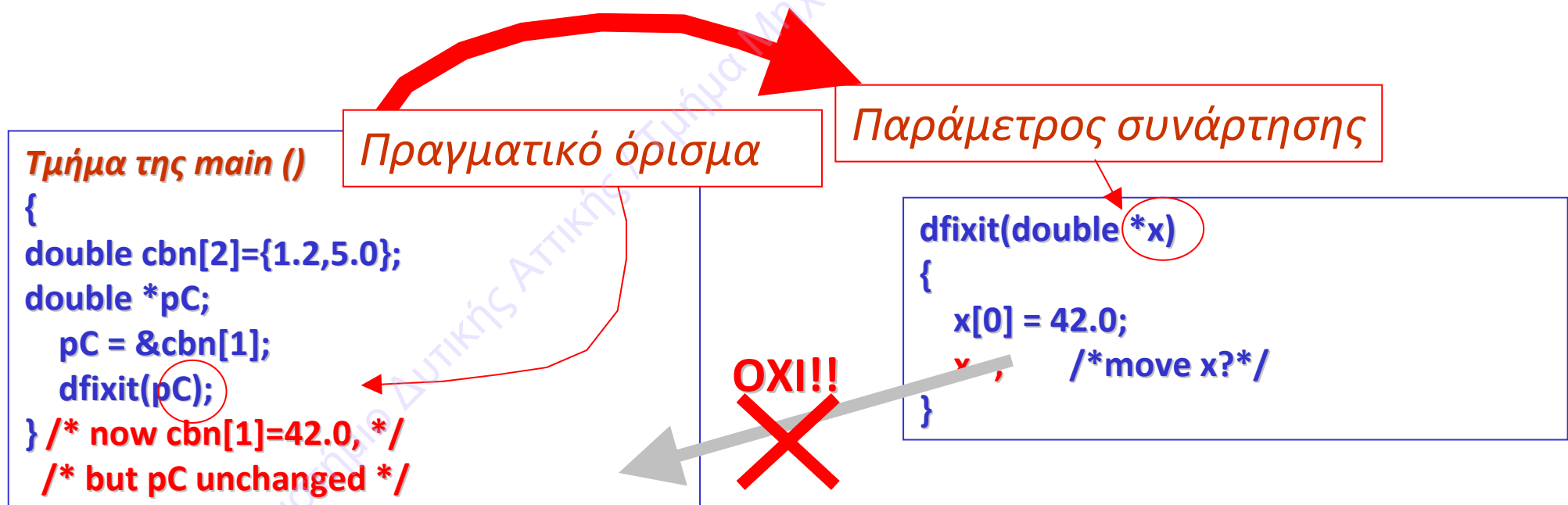
Τμήμα της main ()

```
{ /*double-width key */  
int seed0 = 4; seed1 = 89;  
float key2[20];  
    setKeyP(&key2[0], &seed0);  
    setKeyP(&key2[10],&seed1);  
} /* now seed0,seed1 are 0 */
```

```
setKeyP(float *pK, int *pS)  
{ /* 10 seeded random #s */  
int i;  
    srand(*pS); /* seed... */  
    for(i=0; i<10; i++)  
        pK[i] = rand();  
    *pS=0; /* clear it */  
}
```

Συναρτήσεις και Δείκτες

Οι πραγματικές παράμετροι που είναι δείκτες αντιγράφουν μία **διεύθυνση στις** παραμέτρους της συνάρτησης, αλλά εάν αλλαχθεί η παράμετρος στη συνάρτηση (δηλ. η διεύθυνση) **ΔΕ** θα αλλαχθεί η πραγματική παράμετρος !!



Συναρτήσεις με τύπο επιστροφής δείκτη

Στις συναρτήσεις των οποίων ο τύπος επιστροφής είναι δείκτης, η δήλωση της συνάρτησης μοιάζει:

```
char *findNextVowel(char *str);
```

→!!ΚΙΝΔΥΝΟΣ!!←

- Όλες οι μεταβλητές της συνάρτησης είναι τοπικές και προσωρινές. Μπορεί να **εξαφανισθούν** όταν φύγουμε από τη συνάρτηση.
- Μην επιστρέφετε δείκτες σε μη ορισμένες μεταβλητές!
- Να μην επιστρέφετε ποτέ δείκτες ΕΚΤΟΣ εάν χρησιμοποιείτε τη λέξη κλειδί "static".

Παράδειγμα *static*

Τμήμα της main()

```
{  
    float* pKey;  
  
    pKey = setKey(0);  
    printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);  
    ... (κώδικας που δεν αλλάζει τον pKey) ...  
    printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);  
}
```

ΣΦΑΛΜΑ

```
float* setKey(int s) /* make a cryptographic key */  
{  
    float keep[10];  
    int i;  
    srand(s); /* set rand's seed */  
    for(i=0; i<10; i++) keep[i] = rand();  
    return(keep);  
}
```


Παράδειγμα *static*

Τμήμα της *main()*

```
{  
    float* pKey;  
  
    pKey = setKey(0);  
    printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);  
    ... (code that doesn't change pKey) ...  
    printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);  
}
```

ΣΦΑΛΜΑ

```
float* setKey(int s) /* make a cryptographic key */  
{  
    float keep[10];  
    int i;  
    srand(s); /* set rand's seed */  
    for(i=0; i<10; i++) keep[i] = rand();  
    return(keep);  
}
```

ΔΙΟΤΙ:

Μετά το πέρας της συνάρτησης, η τοπική μεταβλητή *i* και ο πίνακας *keep* είναι ακαθόριστοι

Παράδειγμα *static*

Τμήμα της *main()*

```
{  
    float* pKey;  
  
    pKey = setKey(0);  
    printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);  
    ... (code that doesn't change pKey) ...  
    printf("keys 0,7 are %d,%d.\n",pKey[0],pKey[7]);  
}
```

ΣΩΣΤΟ

```
float* setKey(int s) /* make a cryptographic key */  
{  
    static float keep[10];  
    int i;  
    srand(s);          /* set rand's seed */  
    for(i=0; i<10; i++) keep[i] = rand();  
    return(keep);  
}
```

Η λέξη κλειδί '**static**' διατηρεί τον πίνακα **keep**, ακόμη και μετά την έξοδο από τη συνάρτηση.

επιστροφή(δείκτης) → → Πάντοτε έλεγξε για *static*!

Χρήση δεικτών σε αλφαριθμητικά

- Μπορούν να εκτελεσθούν πράξεις ακεραίων
- Μπορούν να χρησιμοποιηθούν ως κινητά ονόματα πινάκων.

Ο δείκτης πίνακα (array index) [] λειτουργεί!

$*(arr+n) = arr[n]$

$arr+n = \&arr[n]$

- Οι δείκτες μπορούν να ορίσουν ένα string, όπως ακριβώς το ορίζει ένας πίνακας.

Παράδειγμα: Να δοθεί το πρωτότυπο της συνάρτησης ***strlen()*** (μήκος string).

Λύση: Θεωρούμε ότι το όρισμα που δέχεται η συνάρτηση είναι δείκτης σε χαρακτήρα και η δήλωση διαμορφώνεται ως εξής:

int strlen(char *s);

Το σώμα της συνάρτησης με δείκτες και με πίνακες δίνεται παρακάτω:

```
int strlen(char *s)
{
    char *p=s;
    while (*s!='\0')
        s++;
    return(s-p);
}
```

Η εντολή ***s-p*** εκτελεί αφαίρεση δεικτών και δίνει τον αριθμό των στοιχείων μεταξύ των δύο δεικτών.

Παράδειγμα: Να δοθεί το πρωτότυπο της συνάρτησης **strcpy()** (αντιγραφή ενός string σε ένα άλλο).

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

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

Οι εκδόσεις είναι ισοδύναμες; Δοκιμάστε σε δικά σας παραδείγματα!!

Εύρεση χαρακτήρα: *strchr(str1,ch)*

```
#include <stdio.h>      /* για την printf()      */
#include <string.h>      /* για την strcpy()     */

main()
{
    char msg1[81]={“Hello you!”};
    char *fnd;          /* pointer to char */

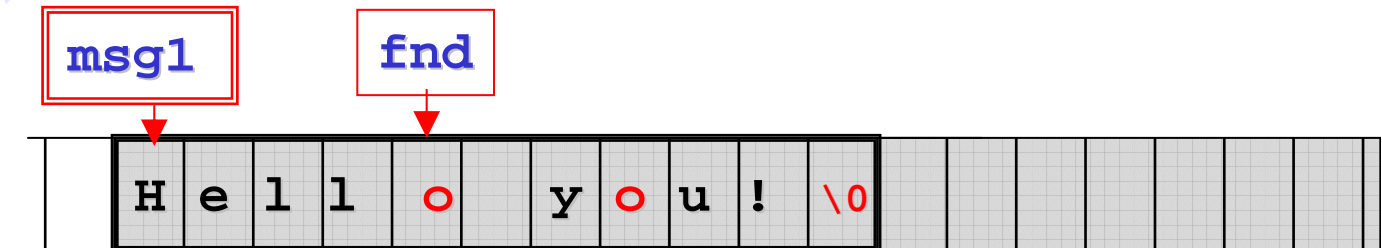
    fnd = strchr(msg1,‘o’);
    printf(“%s\n”, fnd);
}
```

Εύρεση του **char** ‘ch’,
στο string str1.
Επιστροφή δείκτη **char**
στην πρώτη εύρεση.

Αποτέλεσμα:

> o you!

>



Εύρεση string: *strstr(str1, str2)*

```
#include <stdio.h>      /* για την printf() */
#include <string.h>      /* για την strcpy() */

main()
{
    char msg1[81]={"Hello you!"};
    char *fnd;           /* pointer to char */

    fnd = strstr(msg1, "yo");
    printf("%s\n", fnd);
}
```

Εύρεση του **string**
str2 μέσα στο str1;
Επιστροφή δείκτη **char**
στην πρώτη εύρεση.

Αποτέλεσμα:

> you!

>

