

Κεφάλαιο

11

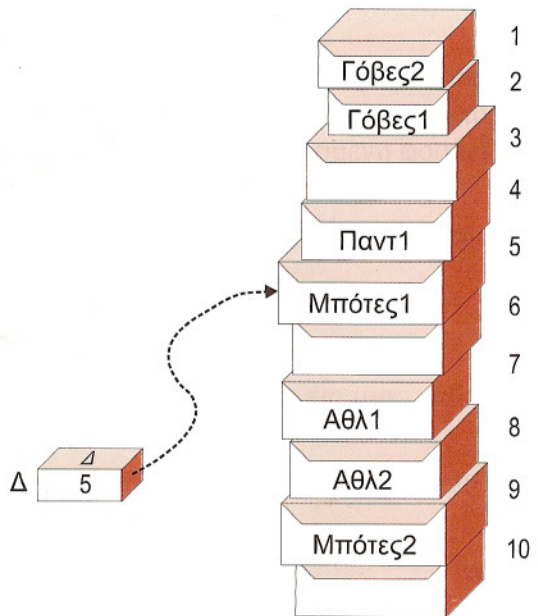
**ΔΕΙΚΤΕΣ**



## ΔΕΙΚΤΕΣ

Οι δείκτες (pointers) είναι ένα από τα πιο "παρεξηγημένα" κεφάλαια των γλωσσών προγραμματισμού. Οι αρχάριοι στον προγραμματισμό συναντούν αρκετή δυσκολία στην κατανόηση και στη χρήση τους, ενώ οι έμπειροι σπαζοκεφαλιάζουν με τις ιδιοτροπίες τους. Η βασική φιλοσοφία των δεικτών, όμως, είναι πολύ απλή.

Ας φανταστούμε κουτιά διαφόρων μεγεθών, το ένα πάνω από το άλλο, τα οποία περιέχουν παπούτσια. Κάποια κουτιά είναι άδεια και κάποια γεμάτα. Κάθε γεμάτο κουτί γράφει στο εξωτερικό του ένα ενδεικτικό όνομα ώστε να καταλαβαίνουμε τι περιέχει. Κάθε κουτί έχει έναν αύξοντα αριθμό (1,2,3 ....) ο οποίος προσδιορίζει τη σειρά του στη στοίβα.



Σε ένα ξεχωριστό κουτί (Δ) βάζουμε τον αύξοντα αριθμό του κουτιού με τα παπούτσια που θα φορέσουμε το βράδυ, π.χ. το 5 αν θέλουμε να βάλουμε το ζευγάρι που βρίσκεται στο κουτί **Μπότες1** (με α/α 5).

Το κουτί Δ δείχνει το κουτί που θα χρησιμοποιήσουμε (το κουτί με α/α 5). Επομένως, το κουτί Δ είναι ένας **δείκτης** ο οποίος δείχνει στο κουτί **Μπότες1**.

Θα μπορούσαμε φυσικά να θυμόμαστε το κουτί που θα χρησιμοποιούσαμε από το όνομά του (**Μπότες1**). Η χρήση του δείκτη Δ, όμως, είναι ένας έμμεσος τρόπος πρόσβασης στο κουτί **Μπότες1**. Έτσι, λοιπόν, για να φορέσουμε τα παπούτσια μας το βράδυ έχουμε δύο επιλογές:

- Να ανοίξουμε το κουτί **Μπότες1** (αν το θυμόμαστε) — ο άμεσος τρόπος.

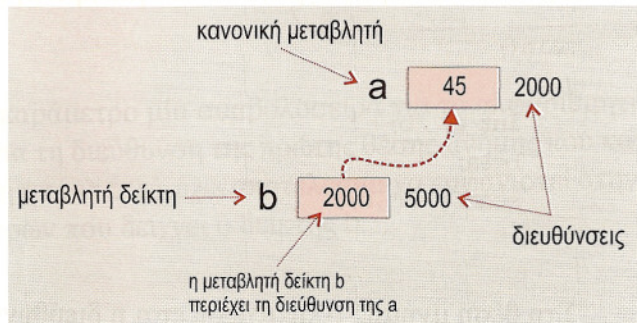
- Να ανοίξουμε το κουτί Δ, να βρούμε ότι το κουτί που μας ενδιαφέρει είναι το κουτί με α/α 5 και να ανοίξουμε το συγκεκριμένο κουτί. Με αυτόν τον τρόπο δε χρειάζεται πια να γνωρίζουμε το όνομα του κουτιού.
- ☞ Στο συγκεκριμένο παράδειγμα το κουτί-δείκτης είναι ένα κουτί το οποίο περιέχει τον α/α ενός άλλου κουτιού.
- ☞ Το μέγεθος του κουτιού-δείκτη **είναι ανεξάρτητο** από το μέγεθος του κουτιού στο οποίο δείχνει, αφού σε κάθε περίπτωση θα περιέχει μόνο τον α/α αυτού του κουτιού.

Το προηγούμενο παράδειγμα δείχνει με απλό και κατανοητό τρόπο τη φιλοσοφία του δείκτη. Στη συνέχεια του κεφαλαίου εξετάζουμε σε βάθος τη χρήση των δεικτών στη C. Η παρουσίαση των δεικτών θα ολοκληρωθεί στο επόμενο κεφάλαιο, αφού γίνει πρώτα κατανοητή η χρήση των πινάκων.

## Μεταβλητές δείκτη (Pointer variables)

Οι μεταβλητές δείκτη έχουν ένα ιδιαίτερο χαρακτηριστικό. Μέσα σε μια μεταβλητή δείκτη καταχωρίζεται η διεύθυνση μιας άλλης μεταβλητής. Η φιλοσοφία μιας μεταβλητής δείκτη φαίνεται εποπτικά στο επόμενο σχήμα.

Η μεταβλητή δείκτη **b** περιέχει την διεύθυνση της μεταβλητής **a**.



## Δήλωση μιας μεταβλητής δείκτη

Μια μεταβλητή δείκτη δηλώνεται όπως οποιαδήποτε άλλη μεταβλητή, με τη διαφορά ότι πριν από το όνομά της πρέπει να υπάρχει ο χαρακτήρας \*. Για παράδειγμα, η

```
int a, *b;
```



δηλώνει μια μεταβλητή **a** τύπου **int** και μια **μεταβλητή δείκτη b**.

Το γεγονός ότι η συγκεκριμένη μεταβλητή δείκτη δηλώθηκε ως τύπου **int** προϋποθέτει ότι η μεταβλητή **b** θα περιέχει τη διεύθυνση μιας μεταβλητής τύπου **int**.

Όπως θα δούμε αργότερα, ο τύπος της θέσης μνήμης στην οποία "δείχνει" μια μεταβλητή δείκτη, παίζει καθοριστικό ρόλο στην αριθμητική των δεικτών.

Προς το παρόν αρκεί να έχουμε υπόψη ότι μια **μεταβλητή δείκτη** περιέχει τη διεύθυνση μιας άλλης μεταβλητής.

☞ Το μέγεθος μιας μεταβλητής δείκτη **δεν** εξαρτάται από τον τύπο της θέσης μνήμης στην οποία "δείχνει" η μεταβλητή. Όλες οι μεταβλητές δείκτη, ανεξάρτητα με την πρόταση δήλωσης με την οποία δηλώθηκαν (**int**, **float** κ.λπ.) έχουν το ίδιο ακριβώς μέγεθος.

☞ Το μέγεθος (σε byte) μιας μεταβλητής δείκτη διαφέρει ανάλογα με την έκδοση της C, το λειτουργικό σύστημα που χρησιμοποιείται, αλλά και τον τύπο του Η/Υ. Σε κάθε περίπτωση, ο τελεστής **sizeof** μπορεί να μας φανερώσει το μέγεθος των μεταβλητών δείκτη στο σύστημα που εργαζόμαστε.

Τα επόμενα παραδείγματα δίνουν μια πρώτη γεύση από τη χρήση των δεικτών.

```
main()
{
    int a, *b;
    b=&a;
    scanf ("%d", b);
}
```

Στη θέση μνήμης **b** καταχωρίζεται η διεύθυνση της μεταβλητής **a**.

Η **scanf ()** περιμένει ως παράμετρο τη διεύθυνση μιας θέσης μνήμης, που είναι στη συγκεκριμένη περίπτωση το περιεχόμενο της **b** (και είναι η διεύθυνση της **a**).

Η **scanf ()** θα περιμένει να πατηθεί ένας αριθμός και θα τον βάλει στη θέση μνήμης με διεύθυνση την τιμή της παραμέτρου της. Δηλαδή στη μεταβλητή **a**.

Στο παραπάνω παράδειγμα, είτε χρησιμοποιηθεί η πρόταση

```
scanf("%d", b);
```

είτε η πρόταση

```
scanf("%d", &a);
```

το αποτέλεσμα θα είναι το ίδιο.

Ας θυμηθούμε ότι μια συμβολοσειρά έχει ως τιμή τη διεύθυνση της πρώτης θέσης μνήμης που καταλαμβάνει το σύνολο χαρακτήρων της (Κεφάλαιο 5).

Μπορούμε να αναθέσουμε σε μια μεταβλητή δείκτη την τιμή μιας συμβολοσειράς και να αναφερόμαστε στη συμβολοσειρά χρησιμοποιώντας τη μεταβλητή δείκτη.

Στο επόμενο παράδειγμα, η τιμή της συμβολοσειράς "Δείκτες" (που είναι η διεύθυνση της πρώτης θέσης μνήμης που καταλαμβάνει το σύνολο χαρακτήρων), καταχωρίζεται στη μεταβλητή δείκτη `b`.

```
main()
{
    char *b;
    b="Δείκτες";
    printf(b);
}
```

Δείκτες

Η `printf()` δέχεται ως παράμετρο μία συμβολοσειρά για το αλφαριθμητικό μορφοποίησης, δηλαδή για τη διεύθυνση της πρώτης θέσης μνήμης που κατέχει. Επομένως, η πρόταση `printf(b)` έχει αποτέλεσμα να εμφανιστεί στην οθόνη το σύνολο χαρακτήρων που δείχνει ο δείκτης `b`.

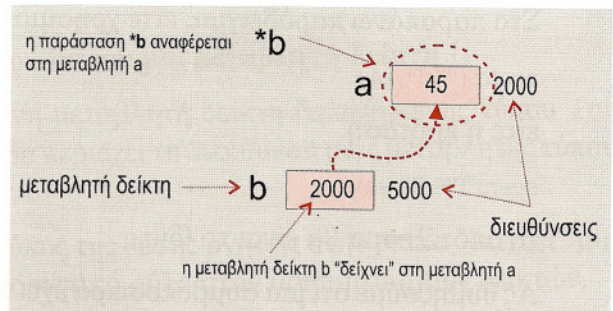
## Οι τελεστές & και \*

Μέχρι στιγμής έχουμε γνωρίσει τον τελεστή `&`, ο οποίος αποδίδει τη διεύθυνση μιας μεταβλητής (θέσης μνήμης).

Γνωρίζουμε ότι μπορούμε να καταχωρίσουμε τη διεύθυνση μιας μεταβλητής σε μια μεταβλητή δείκτη.

Ο τελεστής `*` μας δίνει τη δυνατότητα να έχουμε πρόσβαση στη θέση μνήμης στην οποία "δείχνει" ένας δείκτης.

Αν θεωρήσουμε το παράδειγμα του σχήματος, η μεταβλητή δείκτη `b` περιέχει τη διεύθυνση της μεταβλητής `a` (π.χ. με μία πρόταση `b=&a`).



Χρησιμοποιώντας τον τελεστή `*` μπορούμε να έχουμε πρόσβαση στην μεταβλητή `a` μέσω του δείκτη `b`. Για παράδειγμα, η πρόταση

`*b=100;`

θα καταχωρίσει το 100 στη μεταβλητή `a`.

Ο τελεστής `*` εφαρμόζεται **μόνο** σε μεταβλητές δεικτών και αποδίδει τη μεταβλητή στην οποία δείχνει ο δείκτης<sup>7</sup>.

Στο παρακάτω παράδειγμα γίνεται εμφανής η λογική των δεικτών και η χρήση των τελεστών `&` και `*`.

```
main()
{
    int a,b,*c;
    c=&a;
    a=25;
    printf("To a είναι=%d\n",a);
    printf("To *c είναι=%d\n",*c);
    *c=44;
    printf("To a είναι=%d\n",a);
    printf("To *c είναι=%d\n",*c);
}
```

To a είναι=25  
 To \*c είναι=25  
 To a είναι=44  
 To \*c είναι=44

<sup>7</sup> Ο τελεστής `*` είναι ο ίδιος με τον τελεστή του πολλαπλασιασμού. Η C αντιλαμβάνεται τη διαφορετική χρήση του στην περίπτωση που ακολουθεί μια μεταβλητή δείκτη.



Παρατηρούμε ότι είναι δυνατό να αλλάξουμε την τιμή της μεταβλητής στην οποία δείχνει ένας δείκτης. Έτσι, στο προηγούμενο πρόγραμμα, με την πρόταση:

```
*c=44;
```

το 44 δεν καταχωρίζεται στη μεταβλητή `c` αλλά στη μεταβλητή που δείχνει η `c`, δηλαδή στην `a`.

Οι προτάσεις:

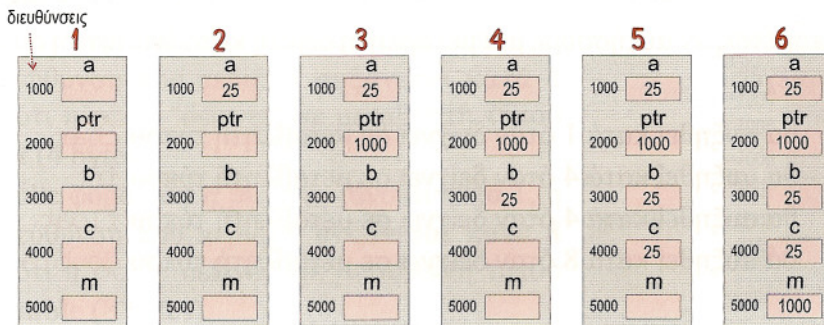
```
a=44;
```

```
*c=44;
```

είναι ισοδύναμες.

Το επόμενο σχηματικό παράδειγμα συμβάλλει στην κατανόηση της χρήσης αυτών των δύο τελεστών. Υποθέτουμε ότι οι διευθύνσεις των μεταβλητών του παραδείγματος είναι οι αριθμοί που εμφανίζονται αριστερά τους.

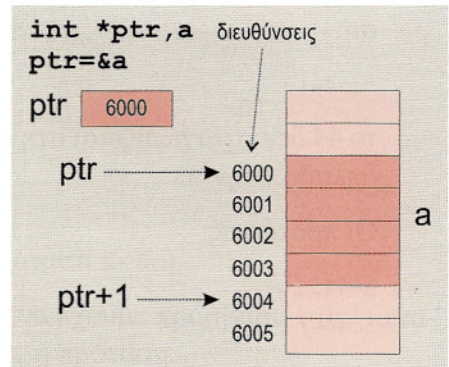
1. `int a, *ptr, b, c, *m;` [δηλώνει 5 μεταβλητές]
2. `a=25;` [βάζει το 25 στην `a`]
3. `ptr = &a;` [βάζει στην `ptr` τη διεύθυνση της `a`]
4. `b = a;` [βάζει στην `b` την τιμή της `a`]
5. `c = *ptr;` [βάζει στη `c` το περιεχόμενο της θέσης μνήμης όπου δείχνει η `ptr`]
6. `m = ptr;` [βάζει στο `m` το περιεχόμενο της `ptr`]



## Αριθμητική των δεικτών

Οι δείκτες ακολουθούν τη δική τους λογική στις αριθμητικές πράξεις, καθόλου όμως παράδοξη αν αναλογιστούμε τη βαθύτερη φιλοσοφία τους.

Ας θεωρήσουμε μια μεταβλητή δείκτη με όνομα `ptr`, η οποία δηλώνεται στο επόμενο πρόγραμμα μαζί με μια μεταβλητή `a`. Υποθέτουμε ότι η μεταβλητή `a` έχει διεύθυνση 6000.



```
int a, *ptr;
```

```
ptr=&a;      [η μεταβλητή ptr θα πάρει την τιμή 6000]
```

```
ptr=ptr+1;   [η μεταβλητή ptr θα πάρει την τιμή 6004]
```

```
++ptr;       [η μεταβλητή ptr θα πάρει την τιμή 6008]
```

```
ptr=ptr+3;   [η μεταβλητή ptr θα πάρει την τιμή 6020]
```

Το παράδοξο είναι ότι ενώ αυξάνουμε τη μεταβλητή `ptr` κατά 1, αυτή αυξάνεται κατά 4 και όταν την αυξήσουμε κατά 3 αυτή αυξάνεται κατά 12!

Πίσω από αυτό το παράδοξο κρύβονται κάποια κρυφά μυστικά των δεικτών.

Μια μεταβλητή δείκτη δείχνει σε μια μεταβλητή συγκεκριμένου τύπου. Αν αυξήσουμε τη μεταβλητή δείκτη κατά ένα, τότε αυτή θα πάρει τέτοια τιμή ώστε να δείχνει στην επόμενη διεύθυνση του ίδιου τύπου.

Επομένως αν αυξήσουμε μια μεταβλητή δείκτη κατά ένα τότε η διεύθυνση που περιέχει:

- θα αυξηθεί κατά 1 όταν δείχνει σε μεταβλητή τύπου `char`
- θα αυξηθεί κατά 4 όταν δείχνει σε μεταβλητή τύπου `int`
- θα αυξηθεί κατά 4 όταν δείχνει σε μεταβλητή τύπου `float`
- θα αυξηθεί κατά 8 όταν δείχνει σε μεταβλητή τύπου `double`



Θα επανέλθουμε στην αριθμητική των δεικτών αργότερα, όταν θα συναντήσουμε και άλλους τύπους δεδομένων όπως π.χ. οι δομές.

**ΠΡΟΣΟΧΗ:** Σε μια μεταβλητή δείκτη δεν μπορούμε να αναθέσουμε οποιαδήποτε αριθμητική τιμή. Μια μεταβλητή δείκτη μπορεί να πάρει τιμή **μόνο** από παράσταση που αποδίδει διεύθυνση μεταβλητής.

`ptr=&a;` [αποδεκτή τιμή — το `&a` αποδίδει διεύθυνση]

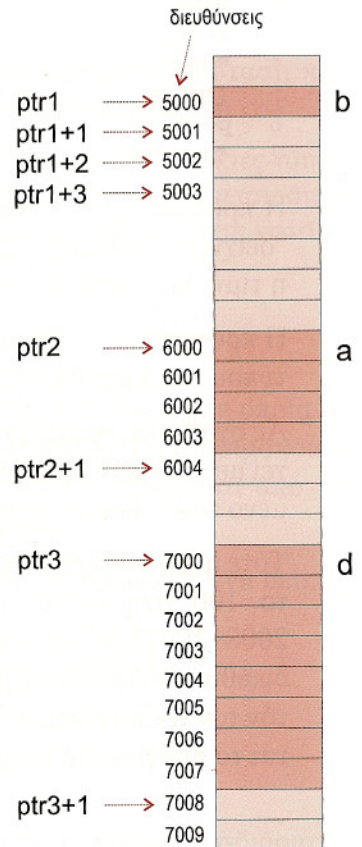
`ptr=1000;` [μη αποδεκτή τιμή — το 1000 είναι `int`]


`ptr=a;` [μη αποδεκτή τιμή — η `a` αποδίδει `int`]

Ας θεωρήσουμε τρεις μεταβλητές `b`, `a`, και `d` με διευθύνσεις 5000, 6000, και 7000 αντίστοιχα. Δηλώνονται επίσης και τρεις μεταβλητές δείκτη, στους οποίους ανατίθενται οι διευθύνσεις των προηγούμενων μεταβλητών:

```
char b, *ptr1;
int a, *ptr2;
double d, *ptr3;
ptr1=&b;
ptr2=&a;
ptr3=&d;
```

- Κάθε φορά που η `ptr1` αυξάνεται κατά 1, η διεύθυνση που περιέχει **αυξάνεται κατά 1** διότι η `ptr1` "δείχνει" σε μεταβλητή τύπου `char` (1 byte).
- Κάθε φορά που η `ptr2` αυξάνεται κατά 1, η διεύθυνση που περιέχει **αυξάνεται κατά 4** διότι η `ptr2` "δείχνει" σε μεταβλητή τύπου `int` (4 bytes).
- Κάθε φορά που η `ptr3` αυξάνεται κατά 1, η διεύθυνση που περιέχει **αυξάνεται κατά 8** διότι η `ptr3` "δείχνει" σε μεταβλητή τύπου `double` (8 bytes).



 ΠΡΟΣΟΧΗ, οι τελεστές ++ και -- έχουν μεγαλύτερη προτεραιότητα από τον \*. Έτσι, η παράσταση **\*ptr++** δεν σημαίνει ότι αυξάνεται η τιμή της μεταβλητής στην οποία δείχνει ο δείκτης **ptr**, αλλά απλά αναφέρεται στη θέση μνήμης στην οποία δείχνει ο **ptr** αφού αυξηθεί κατά ένα (σύμφωνα πάντα με την αριθμητική των δεικτών). Αν θέλαμε όντως να αυξήσουμε κατά 1 την τιμή της μεταβλητής στην οποία έδειχνε ο **ptr**, τότε η παράσταση θα έπρεπε να ήταν **(\*ptr)++**.

Αν υποθέσουμε π.χ. ότι η μεταβλητή **a** έχει διεύθυνση 2000, στο διπλανό πίνακα φαίνονται τα περιεχόμενα των **p** και **a** μετά από κάθε πρόταση του παρακάτω κώδικα:

```
1 int a, *p;
2 p=&a;
3 a=44;
4 *p=9;
5 (*p)++;
6 p++;
7 *p+=100;
```

	p	a
1		
2	2000	
3	2000	44
4	2000	9
5	2000	10
6	2004	10
7	2008	10

100 2008

Η πρόταση **(\*p)++** σημαίνει: αύξησε κατά ένα τη θέση μνήμης στην οποία "δείχνει" ο δείκτης **p**. Στη συγκεκριμένη περίπτωση ο **p** "δείχνει" στην **a**, οπότε η τιμή της **a** από 9 θα γίνει 10.

Η πρόταση **p++** θα αυξήσει το περιεχόμενο του δείκτη **p** κατά 4 (αφού είναι τύπου **int**) και θα πάρει την τιμή 2004.

Ας σταθούμε όμως λίγο στην τελευταία πρόταση **\*p+=100**. Ο τελεστής ++ έχει μεγαλύτερη προτεραιότητα από τον \*, οπότε πρώτα θα αυξηθεί το περιεχόμενο του **p** και θα γίνει 2008.

Τότε η παράσταση θα είναι ισοδύναμη με την **\*(2008)=100** που σημαίνει "βάλε το 100 στη διεύθυνση 2008" —για την ακρίβεια στις θέσεις μνήμης 2008~20011 δεδομένου ότι το 100, ως **int**, χρειάζεται 4 byte. Αυτή η θέση με διεύθυνση 2008 δεν έχει καμία σχέση με τη μεταβλητή **a** και προφανώς με αυτόν τον τρόπο είναι ενδεχόμενο να καταστραφούν άλλα δεδομένα που χειρίζεται το πρόγραμμα μας.

## Δείκτες τύπου Void


Ένας δείκτης μπορεί να είναι και τύπου `void`:

```
void *ptr;
```

Αυτό σημαίνει ότι ο συγκεκριμένος δείκτης **δεν** δείχνει σε δεδομένα συγκεκριμένου τύπου, άρα **δεν** μπορούμε να εφαρμόσουμε αριθμητικούς τελεστές σε αυτόν. Μπορούμε όμως να του αναθέσουμε κανονικά τιμή όπως και στους υπόλοιπους δείκτες.

```
int a;
void *ptr;
ptr=&a;
```

Στο παραπάνω παράδειγμα δεν θα μπορούσαμε να έχουμε παράσταση π.χ. `ptr++`; επειδή η C δεν γνωρίζει πόσο πρέπει να αυξήσει τον δείκτη `ptr`.

 Αρκετές συναρτήσεις βιβλιοθήκης της C επιστρέφουν ως τιμή δείκτες τύπου `void`. Όπως θα αναφέρουμε σε επόμενο κεφάλαιο, η τεχνική της μετατροπής τύπου (type casting) χρησιμοποιείται και για τη μετατροπή ενός δείκτη `void` σε δείκτη ενός συγκεκριμένου τύπου δεδομένων.

## Δείκτης NULL

Ένας δείκτης `null` (null pointer) είναι ένας δείκτης με τιμή `NULL`, η οποία ορίζεται μέσα στο `stdio.h`. Η τιμή `NULL` είναι πρακτικά 0 αλλά χρησιμοποιείται στις περιπτώσεις των δεικτών αντί για το 0, επειδή εξαρτάται από το μοντέλο μνήμης που χρησιμοποιείται στο σύστημα μας.

Παρόλο που φαίνεται το ίδιο, η ανάθεση του 0 αντί της `NULL` σε ένα δείκτη μπορεί να έχει καταστροφικά αποτελέσματα.

Στις επόμενες προτάσεις ανατίθεται στο δείκτη `p` η τιμή `NULL`:

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

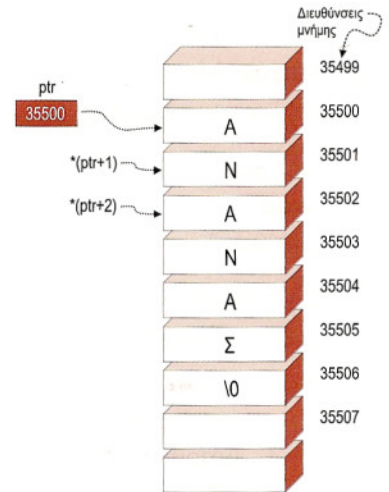
 ΠΡΟΣΟΧΗ: Ένας δείκτης, αμέσως μετά τη δήλωση του, έχει **απροσδιόριστη** τιμή και όχι 0 (`NULL`).



## Παραδείγματα

**Π.1** Το παρακάτω πρόγραμμα μετράει τους χαρακτήρες 'A' που βρίσκονται μέσα στη συμβολοσειρά "ANANAS".

```
main()
{
    char *ptr, ch;
    int c=0;
    ptr="ANANAS";
    while(*ptr != '\0')
    {
        if(*ptr == 'A') c++;
        ptr++;
    }
    printf("Η λέξη έχει %d A\n", c);
}
```



Αρχικά καταχωρίζεται στη μεταβλητή **ptr** η διεύθυνση της συμβολοσειράς "ANANAS" (στο παράδειγμα, η 35500). Η παράσταση **\*ptr** αναφέρεται στη θέση μνήμης όπου δείχνει κάθε φορά ο δείκτης **ptr**. Η παράσταση **ptr++** αυξάνει το δείκτη ώστε να δείχνει στην επόμενη θέση. Αρχικά ο **ptr** δείχνει στο πρώτο 'A', μετά στο 'N', μετά στο δεύτερο 'A' κ.ο.κ. Η διαδικασία θα σταματήσει μόλις το **ptr** δείξει στο '\0', το οποίο σηματοδοτεί το τέλος του συνόλου χαρακτήρων. Η μεταβλητή **c** χρησιμοποιείται για να μετράει τις θέσεις που περιέχουν τον χαρακτήρα 'A'.

**Π.2** Το επόμενο πρόγραμμα εμφανίζει τους χαρακτήρες που βρίσκονται μέσα στη συμβολοσειρά "ANANAS" με αντίστροφη σειρά ("SANANA").

```
main()
```

```
{
```

```
    char *ptr,*arxi,ch;
```

```
    int c=0;
```

```
    ptr="ANANAS";
```

```
    arxi=ptr;
```

```
    while(*ptr != '\0')
```

```
    {
```

```
        ptr++;
```

```
    }
```

```
    ptr--;
```

```
    while(ptr >= arxi)
```

```
    {
```

```
        putchar(*ptr);
```

```
        ptr--;
```

```
    }
```

```
}
```

Στον δείκτη arxi καταχωρίζουμε τα περιεχόμενα του ptr ώστε να μη χαθεί η αρχή της συμβολοσειράς.

Αυξάνουμε το ptr μέχρι να δείχνει στο τέλος των χαρακτήρων (το '\0').

Μειώνουμε τον ptr κατά 1 ώστε να δείχνει τον τελευταίο χαρακτήρα πριν από το '\0'.

Εμφανίζουμε ένα-ένα χαρακτήρα μέχρι το ptr να δείξει πάλι στην αρχή των χαρακτήρων.

## Ανασκόπηση Κεφαλαίου 11

- Μια μεταβλητή δείκτη είναι μια θέση μνήμης η οποία περιέχει τη διεύθυνση μιας άλλης θέσης μνήμης.
- Μια μεταβλητή δείκτη δηλώνεται όπως μία οποιαδήποτε άλλη μεταβλητή, με τη διαφορά ότι πριν από το όνομά της πρέπει να υπάρχει ο χαρακτήρας \*. π.χ. η `int a, *b` δηλώνει μία ακέραια μεταβλητή `a` και μία μεταβλητή δείκτη με όνομα `b`.
- Μέσα σε μια μεταβλητή δείκτη μπορούμε να καταχωρίσουμε μόνο τη διεύθυνση μιας θέσης μνήμης. Για παράδειγμα, η `b=&a` καταχωρίζει στην `b` τη διεύθυνση της μεταβλητής `a`.
- Όταν η μεταβλητή δείκτη `b` περιέχει τη διεύθυνση μιας άλλης μεταβλητής `a`, τότε λέμε ότι ο δείκτης `b` "δείχνει" στη μεταβλητή `a`.
- Χρησιμοποιώντας τον τελεστή \* μπορούμε να έχουμε πρόσβαση μέσω μιας μεταβλητής δείκτη στη θέση στην οποία "δείχνει" η μεταβλητή. Μπορούμε δηλαδή να αναφερθούμε στην `a` μέσω της `b`. Για παράδειγμα, η `*b=5` καταχωρίζει το 5 στη μεταβλητή `a` (το `*b=5` είναι ισοδύναμο με το `a=5`).
- Όταν αυξάνουμε κατά 1 μια μεταβλητή δείκτη, τότε αυτή αυξάνεται κατά τόσες μονάδες όσα είναι τα byte του τύπου δεδομένων στον οποίο δηλώθηκε. Για παράδειγμα, η `b++` αυξάνει το `b` κατά 4 επειδή ο τύπος `int` (στον οποίο δηλώθηκε η `b`) έχει μέγεθος 4 byte. Το αντίστοιχο γίνεται και στη μείωση.
- Κάθε μονάδα που προστίθεται σε ένα δείκτη τύπου `char` αυξάνει το δείκτη κατά 1, σε ένα δείκτη τύπου `int` κατά 4, σε ένα δείκτη τύπου `float` επίσης κατά 4, και σε ένα δείκτη τύπου `double` κατά 8.
- Ένας δείκτης `null` είναι ένας δείκτης με τιμή `NULL` η οποία ορίζεται μέσα στο `stdio.h`. Ένας δείκτης με τιμή `NULL` δεν "δείχνει" πουθενά.



## Ασκήσεις Κεφαλαίου 11

**11.1** Τι θα περιέχουν οι μεταβλητές *a*, *b*, και *c* μετά το τέλος του επόμενου προγράμματος: ★

```
main()
{
    int a,b,c,*m,*p;
    a=100;
    b=50;
    m=&a;
    p=&b;
    c=*p + *m;
    (*p)++;
    p=m;
    (*p)--;
}
```

**11.2** Τι κάνει το επόμενο πρόγραμμα: ★

```
main()
{
    char *p;
    int a=5,b=10;
    p="a,b=%d,%d\n";
    printf(p,a,b);
}
```

Τι θα έκανε αν αντί για,

```
printf(p,a,b);
```

είχαμε

```
printf(p+4,a,b);
```

**11.3** Με δεδομένο το επόμενο πρόγραμμα:

```
main()
{
    char *p;
    p="αρνάκι άσπρο και παχύ";
    .....
}
```

Να συμπληρωθεί κατάλληλα, ώστε να ζητάει ένα γράμμα και μετά να μετράει πόσες φορές υπάρχει το γράμμα αυτό στο σύνολο χαρακτήρων "αρνάκι άσπρο και παχύ". ★ ★

#### 11.4 Τι θα εμφανιστεί στην οθόνη από το επόμενο πρόγραμμα; ★ ★

```
main()
{
    char *p;
    p="αρνάκι άσπρο και παχύ";
    while(*p!='\0')
    {
        if(*p==' ')
            putchar('\n');
        else
            putchar(*p);
        p++;
    }
}
```

#### 11.5 Με δεδομένο το επόμενο πρόγραμμα:

```
main()
{
    int a,b,c,*p1,*p2,*p3;
    p1=&a;
    p2=&b;
    p3=&c;
    .....
}
```

Να συμπληρωθεί κατάλληλα, ώστε να ζητάει με τη `scanf()` δύο αριθμούς, να τους καταχωρίζει στις μεταβλητές `a` και `b` αντίστοιχα, να υπολογίζει το άθροισμα των `a` και `b`, να το καταχωρίζει στη `c`, και να εμφανίζει το περιεχόμενο της `c` στην οθόνη. Σε καμία όμως από τις νέες προτάσεις, δεν πρέπει να εμφανίζονται τα ονόματα των μεταβλητών `a`, `b`, και `c`. ★ ★

- 11.6** Με δεδομένη την επόμενη πρόταση, συμπληρώστε τον πίνακα με την αντίστοιχη ερμηνεία: ★

```
int a,b,c,*p,*m;
```

Πρόταση	Ερμηνεία
<code>p=&amp;a</code>	
<code>*p=23</code>	
<code>m=&amp;b</code>	
<code>c=*p + *m</code>	
<code>m=p</code>	
<code>p++</code>	
<code>(*p)++</code>	
<code>m=120</code>	

- 11.7** Τι θα εμφανιστεί στην οθόνη από το επόμενο πρόγραμμα; ★ ★ ★

```
main()
{
    char *p,*m;
    p="αρνάκι άσπρο και παχύ";
    m=p;
    while(*p!='\0') p++;
    --p;
    while(p>=m) putchar(*(p--));
}
```



**11.8** Στον επόμενο πίνακα σημειώστε τα περιεχόμενα κάθε μεταβλητής μετά από κάθε μια από τις αριθμημένες προτάσεις: ★

```

int a,b,c=0,*ptr,*m;
a=b=10;
1 ptr=&a;
  *ptr=34;
2 m=&b;
3 c=*ptr + *m;
  ptr=m;
4 *ptr=100;
5 m++;
    
```

1	2	3	4	5
<div> <div>Διευθύνσεις</div> <div> <div>a</div> <div>ptr</div> <div>b</div> <div>c</div> <div>m</div> </div> <div> <div>1000</div> <div>2000</div> <div>3000</div> <div>4000</div> <div>5000</div> </div> </div>	<div> <div>a</div> <div>ptr</div> <div>b</div> <div>c</div> <div>m</div> </div>	<div> <div>a</div> <div>ptr</div> <div>b</div> <div>c</div> <div>m</div> </div>	<div> <div>a</div> <div>ptr</div> <div>b</div> <div>c</div> <div>m</div> </div>	<div> <div>a</div> <div>ptr</div> <div>b</div> <div>c</div> <div>m</div> </div>

**11.9** Ποια από τα επόμενα αληθεύουν: ★

- ☐ Ένας δείκτης μπορεί να περιέχει **μόνο** τη διεύθυνση μιας θέσης μνήμης.
- ☐ Ο τελεστής \* χρησιμοποιείται για να έχουμε πρόσβαση σε μια θέση μνήμης μέσω ενός δείκτη ο οποίος περιέχει τη διεύθυνσή της.
- ☐ Αν εφαρμόσουμε τον τελεστή ++ σε μια μεταβλητή δείκτη, αυτή αυξάνεται κατά 1.
- ☐ Το μέγεθος μιας μεταβλητής δείκτη εξαρτάται από το σύστημα στο οποίο εκτελείται το πρόγραμμά μας.
- ☐ Μια μεταβλητή δείκτη **char** και μια μεταβλητή δείκτη **int** έχουν διαφορετικό μέγεθος.