ΕΙΣΑΓΩΓΗ ΣΤΟ ΔΟΜΗΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

Τέταρτη Διάλεξη
ΠΙΝΑΚΕΣ

Έστω ότι έχουμε το εξής πρόβλημα: εισάγουμε μια σειρά αριθμών από το πληκτρολόγιο και θέλουμε να βρούμε τον μέγιστο και τον ελάχιστο. Γενικά, είναι εύκολο να υλοποιηθεί ένα πρόγραμμα για το σκοπό αυτό (βλ. και τρίτη σειρά ασκήσεων). Τι γίνεται όμως, αν, μετά από την εύρεση του μεγίστου και ελαχίστου, θέλουμε να υποβάλουμε τους ίδιους αριθμούς και σε περαιτέρω επεξεργασία;

- Μία λύση θα ήταν να τους ξαναβάζαμε με το χέρι – απαράδεκτη εκτός από πολύ μικρό αριθμό δεδομένων και πολύ μικρά προγράμματα, μάλλον χωρίς ιδιαίτερη χρησιμότητα.
- Δεύτερη λύση είναι το πρόγραμμα να γράψει τους αριθμούς σε ένα αρχείο και να το ξαναδιαβάσει όποτε χρειαστεί. Αυτό έχει τρία προβλήματα: η σειριακή ανάγνωση μας επιβάλλει και μια σειρά χειρισμού των αριθμών, αν έχουμε χιλιάδες αριθμούς "τρώμε" δίσκο και η ανάγνωση και εγγραφή είναι συγκριτικά χρονοβόρες και για μαζικό χειρισμό δεδομένων θα επιβραδύνει το πρόγραμμα.
- Η τρίτη λύση είναι η χρήση πινάκων (arrays). Οι πίνακες είναι ειδική περίπτωση ή η πρώτη απλή περίπτωση τωνλεγόμενων δομών δεδομένων που θα μελετήσουμε στη γενικότητά τους σε επόμενο μάθημα.

Πίνακας (array) είναι ένα σύνολο μεταβλητών *του ιδίου τύπου* που έχουν κοινό όνομα. Αυτές διακρίνονται η μία από την άλλη, από έναν ακέραιο αριθμό που ονομάζεται αριθμοδείκτης ή απλά δείκτης του πίνακα.

Αν ένας πίνακας έχει N στοιχεία, τότε ο αριθμοδείκτης κυμαίνεται από **μηδέν** μέχρι N-1.

Τρόπος δήλωσης και γραφής:

Δήλωση ακέραιου πίνακα με δέκα στοιχεία:

```
int a[10];
```

Απόδοση της τιμής 9 στο πρώτο και στο δέκατο στοιχείο του παραπάνω πίνακα:

```
a[0] = a[1] = 9;
```

Τύπος – όνομα – μέγεθος

Είχαμε πει ότι οι μεταβλητές έχουν τύπο και όνομα, ενώ οι συναρτήσεις έχουν τύπο, όνομα και προαιρετική λίστα παραμέτρων με τύπους και ονόματα. Οι πίνακες έχουν τύπο, όνομα και μέγεθος που ισούται με τον αριθμό των στοιχείων τους. Μια απλή μεταβλητή καταλαμβάνει μία θέση μνήμης ή τόσες όσες χρειάζονται για τον τύπο της. Ένας πίνακας καταλαμβάνει το γινόμενο των bytes που αντιστοιχούν στον τύπο επί το μέγεθος του πίνακα. Αυτό το ποσό μνήμης δεσμεύεται αμέσως μόλις γίνει η δήλωση του πίνακα. Τα στοιχεία του πίνακα αποθηκεύονται στη σειρά το ένα μετά το άλλο (αυτό το τονίζουμε επειδή δεν είναι δεδομένο – με τους δείκτες που θα δούμε παρακάτω μπορούμε να κατασκευάσουμε δομές δεδομένων όπου δεν ισχύει απαραίτητα η διαδοχική τοποθέτηση).

Για να βρούμε το μέγεθος σε bytes που καταλαμβάνει μία απλή μεταβλητή μπορούμε να χρησιμοποιήσουμε τον τελεστή sizeof, π.χ. για τη μεταβλητή var γράφουμε: sizeof(var). Το ίδιο μπορούμε να κάνουμε και για έναν τύπο μεταβλητής, π.χ. sizeof(float). Το ίδιο ακριβώς μπορούμε να κάνουμε και για έναν πίνακα. Επιπλέον, αν είμαστε κάπου μέσα σε ένα πολύ μεγάλο πρόγραμμα και θέλουμε να βρούμε επί τόπου το μέγεθος ενός πίνακα (τον αριθμό των στοιχείων του) με όνομα έστω anArray, τότε αρκεί να γράψουμε:

`sizeof(anArray) / sizeof(anArray[0])`.

Σύνηθες λάθος εκτέλεσης: απόπειρα πρόσβασης στοιχείου πίνακα που δεν έχει οριστεί. Π.χ. έχουμε μία δομή for με μια ακέραιη μεταβλητή ελέγχου που χρησιμεύει ως αριθμοδείκτης κάποιου πίνακα και σε κάποια στιγμή λόγω πράξεων που εκτελούνται στα πλαίσια του προγράμματος, ξεπερνά το μέγεθος του πίνακα. Χρειάζεται προσοχή! Εδώ είναι που μπορεί να βοηθήσει ο τελεστής sizeof με τον τρόπο που δείξαμε.

Αρχικοποίηση – όπως και με τις απλές μεταβλητές, δηλαδή είτε στη δήλωση είτε μετά:

- στη δήλωση: `int length[3] = {1, 3, 5};`
- στο κύριο σώμα του προγράμματος, είτε ένα-ένα στοιχείο κατά περίπτωση είτε όλα μαζί συνήθως με κάποια δομή for.

Στην αρχικοποίηση με δήλωση μπορούμε να παραλείψουμε το μέγεθος μέσα στις αγκύλες γιατί ο μεταγλωττιστής θα μετρήσει τα στοιχεία που θα βρει στη λίστα μέσα στα άγκιστρα:

```
int length[] = {1, 3, 5}; /* σωστό! */
```

Αν στη δήλωση ορίσουμε μόνο μερικά στοιχεία, τα υπόλοιπα συμπληρώνονται αυτόματα με 0. Π.χ. float x[5] = {1.2, 3.24, 6.8};

εδώ τα δύο τελευταία είναι ίσα με 0. Αυτό επιτρέπει ένα καλό τέχνασμα για να αρχικοποιούμε μεμιάς μεγάλους πίνακες: αρκεί να αρχικοποιήσουμε ένα στοιχείο του, ως εξής:

```
int big[10000] = {0};
```

Όλα έγιναν αυτόματα μηδέν!

Παραδείγματα με for για άλλες περιπτώσεις για τις οποίες δε μας κάνει το μηδέν ως αρχική τιμή:

```
int i, x[50];

for(i=0; i<50; i++) x[i] = i;
```

Ας δώσουμε διαφορετικές τιμές στα στοιχεία με άρτιο αριθμοδείκτη από αυτά με περιττό:

```
for(i=0; i<25; i++){

   x[2*i] = 1;

   x[2*i+1] = 0;

}
```

Δηλαδή μπορούμε να βάλουμε ακόμη και ολόκληρες (ακέραιες) παραστάσεις για αριθμοδείκτη αρκεί να μη βγαίνουν εκτός ορίων του πίνακα.

Πίνακες χαρακτήρων και αλφαριθμητικά.

Μέχρι τώρα όταν θέλαμε να τυπώσουμε κάτι στην οθόνη καλούσαμε την printf και μέσα βάζαμε κάποια φράση σε " ". Αυτά είναι τα λεγόμενα αλφαριθμητικά (strings). Από την άλλη, είδαμε ότι υπάρχουν και μεταβλητές char που όταν τους αποδίδουμε τιμή πρόκειται για ένα και μόνο χαρακτήρα σε απλά εισαγωγικά ' '. Τι συμβαίνει εδώ; Οι πίνακες μπορούν να είναι και τύπου char. Τα αλφαριθμητικά αποθηκεύονται σαν πίνακες χαρακτήρων που όμως έχουν επιπλέον και έναν "αόρατο" χαρακτήρα στο τέλος, τον λεγόμενο κενό χαρακτήρα '\0' (καμία σχέση με το κενό διάστημα, ' '). Οι πίνακες χαρακτήρων δεν είναι αλφαριθμητικά εφόσον δεν έχουν τον κενό χαρακτήρα στο τέλος. *Ένα αλφαριθμητικό με Ν γράμματα έχει μέγεθος Ν+1 λόγω και του κενού*

χαρακτήρα.

Για να ορίσουμε και αρχικοποίησουμε ένα αλφαριθμητικό, θα ορίσουμε έναν πίνακα τύπου char, αλλά δεν είναι ανάγκη να ορίσουμε το μέγεθός του, διότι ο μεταγλωττιστής θα το αποδώσει αυτόματα:

```
char lala[] = "Kozani city";
```

Τα διπλά εισαγωγικά τονίζουν τη διαφορετική φύση των αλφαριθμητικών από τους γενικούς πίνακες χαρακτήρων.

Σημείωση: για να τυπώσω ένα αλφαριθμητικό αποθηκευμένο σε μεταβλητή με τη συνάρτηση printf χρησιμοποιώ την προδιαγραφή %s, δηλαδή, π.χ.

```
printf("%s", lala);
```

δίνει

```
Kozani city
```

Πολυδιάστατοι πίνακες.

Οι πίνακες που εξετάσαμε μέχρι εδώ είναι μονοδιάστατοι και μοιάζουν περισσότερο με αυτό που λέμε στα μαθηματικά, ανύσματα. Μπορούν όμως να οριστούν και πίνακες όπου η θέση ενός στοιχείου προσδιορίζεται από δύο αριθμοδείκτες. Αυτοί οι πίνακες μοιάζουν με αυτούς που ξέρουμε από τα μαθηματικά και λέμε ότι έχουν διαστατικότητα (dimensionality) δύο, αντίθετα από αυτούς που είδαμε ως εδώ και που έχουν διαστατικότητα ένα. Στην πραγματικότητα, μπορούν να οριστούν πίνακες διαστατικότητας Ν γενικότερα. Αυτοί χρειάζονται Ν αριθμοδείκτες για να προσδιοριστεί η θέση ενός στοιχείου τους. Βλέπουμε ότι τέτοιοι πίνακες για να δηλωθούν χρειάζονται τύπο, όνομα, διαστατικότητα και μέγεθος ανά διάσταση, δηλαδή πλήθος στοιχείων που το γράφουμε σε ξεχωριστές αγκύλες.

Δήλωση:

Δήλωση πίνακα με 5 γραμμές και 4 στήλες.

```
int a[5][4];
```

Κάθε γραμμή είναι στην πραγματικότητα με τη σειρά της ένας μονοδιάστατος πίνακας. Αυτός είναι ο λόγος που γράφουμε a[m][n] και όχι a[m, n] όπως θα κάναμε σε άλλες γλώσσες προγραμματισμού.

Αρχικοποίηση:

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

Μπορούμε να τα γράψουμε και το ένα κάτω από το άλλο για πιο ευανάγνωστα:

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

Αλλά θα μπορούσαν να δοθούν και σα μία μονοκόμματη λίστα {1, 2, 3, 3, 2, 5} και ο μεταγλωττιστής θα ήξερε που να τα βάλει.

Όταν αρχικοποιούμε ταυτόχρονα με τη δήλωση, μπορούμε να αφήσουμε την πρώτη (και μόνο την πρώτη!) διάσταση απροσδιόριστη διότι θα οριστεί αυτόματα με την αρχικοποίηση, π.χ.:

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

Σημείωση 1: πρώτα πάνε οι γραμμές και μετά οι στήλες οι οποίες και μεταβάλλονται πιο γρήγορα (δηλαδή στην πράξη τα στοιχεία είναι αποθηκευμένα στη σειρά με τη λογική [1][1], [1][2],...[1][N],[2][1],[2][2],...[2],[N],...[N][1],[N][2]...[N][N].

Σημείωση 2: μπορούμε να έχουμε και περισσότερες από δύο διαστάσεις. Οι πιο δεξιές μεταβάλλονται γρηγορότερα.

Επεξεργασία: ταιριάζουν ένθετες δομές for. Έτσι, σε προγράμματα που χρησιμοποιούν πίνακες δύο διαστάσεων είναι πολύ συνηθισμένες οι επαναλήψεις της μορφής

```
for (i = 0; i < m; i++)

    for (j = 0; j < n; j++)

        someArray[i][j] = κάποια_παράσταση(i, j);
```

ΔΕΙΚΤΕΣ

Είχαμε δει στην εντολή scanf το σύμβολο & μπροστά από τη μεταβλητή ανάγνωσης. Αυτός είναι ο τελεστής διεύθυνσης που δίνει τη διεύθυνση της μεταβλητής όπου επιδρά. Η διεύθυνση μπορεί να αποδοθεί σε μια μεταβλητή που λέγεται δείκτης (pointer).

Οι μεταβλητές δείκτη ή απλά δείκτες (pointers) είναι μεταβλητές που η τιμή τους είναι ίση με τη διεύθυνση μιας άλλης μεταβλητής.

Οι δείκτες επιταχύνουν την πρόσβαση στα δεδομένα και το χειρισμό τους και προσφέρουν ευελιξία στη διαδικασία ανάπτυξης λογισμικού επιτρέποντας εναλλακτικούς τρόπους απεύθυνσης (addressing).

Οι δείκτες, όπως και οι συνήθεις μεταβλητές, δηλώνονται στην αρχή του προγράμματος ή της συνάρτησης που τους χρησιμοποιεί και με τρόπο εντελώς ανάλογο. Όπως υπάρχουν τύποι μεταβλητών έτσι υπάρχουν και αντίστοιχοι τύποι δεικτών (char, int, float, double κλπ) ανάλογα με το τι τύπου μεταβλητή είναι αυτή της οποίας τη διεύθυνση αποθηκεύει ένας δείκτης.

Για τον ορισμό και γενικότερα το χειρισμό των δεικτών υπάρχουν δύο τελεστές. Τον ένα είδαμε ήδη και είναι ο τελεστής διεύθυνσης &. Ο άλλος είναι το * και είναι ο τελεστής περιεχομένου (ανάλογα με τη χρήση του * ο μεταγλωττιστής καταλαβαίνει αν πρόκειται για τελεστή περιεχομένου ή για σημείο πολλαπλασιασμού). Έτσι, αν x και px είναι μια πραγματική μεταβλητή και ένας δείκτης επίσης πραγματικού τύπου, μπορούμε να γράψουμε

```
px = &x;

x = *px;
```

όπου, η πρώτη σχέση βρίσκει τη διεύθυνση της x (μέσω του τελεστή &) και την αποδίδει στον δείκτη px και η δεύτερη σχέση βρίσκει το **περιεχόμενο** που υπάρχει στη διεύθυνση την αποθηκευμένη στον δείκτη px και την αποδίδει στη μεταβλητή x. Για την πρώτη σχέση λέμε ότι ο δείκτης px **δείχνει** στη μεταβλητή x ενώ η δεύτερη λέγεται dereference (ελληνικά;) του δείκτη px.

Με βάση τα παραπάνω μπορούμε να παρουσιάσουμε αλλά και να εξηγήσουμε καλύτερα τον τρόπο δήλωσης των μεταβλητών δείκτη. Π.χ. για να ορίσουμε έναν δείκτη πραγματικού τύπου px θα γράψουμε:

```
float *px;
```

Αυτή η εντολή διατυπωμένη με λόγια λέει ότι αν πάρουμε το περιεχόμενο της διεύθυνσης που είναι αποθηκευμένη στο px θα προκύψει ένας πραγματικός αριθμός.

Προσοχή! Το όνομα του δείκτη είναι px και *όχι* *px! Ο τελεστής * στη δήλωση μας μπαίνει εκεί

για να μας πει τι θα συμβεί αν πάρουμε το περιεχόμενο της διεύθυνσης px.

Βλέπουμε ότι όπως και οι συνήθεις μεταβλητές έτσι και οι δείκτες έχουν όνομα και τύπο στον οποίο δείχνουν. Εκτός από τους ήδη γνωστούς τύπους δεικτών μπορούν να οριστούν και δείκτες τύπου void που είναι κάτι σαν δείκτες "γενικής χρήσης" χρήσιμοι σε ορισμένες ειδικές περιπτώσεις που θα παρουσιάσουμε αργότερα.

Δείκτες και εντολές απόδοσης.

Ένας δείκτης, όπως και μια απλή μεταβλητή, μπορεί να μπει στο αριστερό σκέλος μιας εντολής απόδοσης για να του αποδοθεί μια διεύθυνση. Είδαμε ήδη ένα παράδειγμα. Μπορεί να εξισωθεί και με έναν άλλο δείκτη. Σε κάθε περίπτωση, αυτό συνεπάγεται αλλαγή του περιεχομένου της μεταβλητής στην οποία δείχνει ο δείκτης.

Δείκτες και πίνακες.

Στη C υπάρχει πολύ στενή σχέση μεταξύ δεικτών και πινάκων. Μπορούμε να δουλέψουμε με πίνακες χρησιμοποιώντας δείκτες αλλά και να αναφερόμαστε σε δείκτες με τον ίδιο τρόπο που αναφερόμαστε σε στοιχεία πινάκων. Αυτό είναι κατανοητό αν σκεφτούμε ότι οι πίνακες είναι ομάδες μεταβλητών τοποθετημένες σε διαδοχικές διευθυνσεις επομένως μπορούμε μεταβάλλοντας κατάλληλα την τιμή μιας μεταβλητής δείκτη που δείχνει κάπου μέσα σε έναν πίνακα να έχουμε πρόσβαση στα διάφορα στοιχεία του. Επίσης μπορούμε να αναφερθούμε σε έναν πίνακα μέσω της διεύθυνσης ενός συγκεκριμένου στοιχείου του και είναι πιο βολικό να διαλέξουμε το πρώτο. Πράγματι, αυτή είναι και η σύμβαση που υιοθετείται από τη γλώσσα C:

Έστω ένα πίνακας float pinax[300] ή κάτι ανάλογο. Τότε το όνομά του, π.χ. pinax είναι **συνώνυμο** με τη θέση (διεύθυνση) του πρώτου στοιχείου του, pinax[0].

Για να αποκτήσουμε πρόσβαση στα στοιχεία του πίνακα, εκτός από τον "κλασσικό" τρόπο με αριθμοδείκτες, pinax[0], pinax[1] κλπ, έχουμε και άλλον ένα που βασίζεται στη λεγόμενη **αριθμητική δεικτών**, δηλαδή μπορούμε να γράψουμε αντίστοιχα *pinax, *(pinax+1), κλπ. Τι θα πει αυτό; Προσθέτοντας 1, 2, κλπ σε ένα δείκτη προχωρούμε στην επόμενη διεύθυνση. Δεν αυξάνουμε το δείκτη κυριολεκτικά κατά 1 αλλά είναι μια συντομογραφία για να πούμε ότι πάμε τόσα bytes πιο πέρα όσα είναι αυτά που χρειάζονται για την αποθήκευση μεταβλητής του τύπου όπου δείχνει ο δείκτης (εδώ float).

Προσοχή! Υπάρχει ένας περιορισμός, ότι οι πίνακες είναι **σταθερές δείκτη** δηλαδή δεν επιτρέπεται να τους αλλάξουμε την τιμή και να αποδώσουμε κάποια άλλη μέσω μιας εντολής απόδοσης. Μόνο αλλάζοντας με τον "κλασσικό" τρόπο ένα-ένα τα στοιχεία του μπορούμε να αλλάξουμε έναν πίνακα!

Πάντως, λόγω της παραπάνω ισοδυναμίας μπορούμε να "αποδώσουμε" έναν ολόκληρο πίνακα σε έναν δείκτη, δηλαδή, π.χ.:

```
float array[10];

float *ptr;

ptr = array;
```

εννοώντας ότι ο δείκτης θα δείχνει στο πρώτο στοιχείο του πίνακα. Η τελευταία παραπάνω εντολή είναι το ίδιο σα να γράφαμε:

```
ptr = &array[0];
```

Η **αριθμητική δεικτών** μου επιτρέπει να αναπαραστήσω με πολλούς τρόπους ένα στοιχείο πίνακα:

`*(ptr+i)` και `array[i]` είναι το ίδιο πράγμα (περιεχόμενο του i+1 στοιχείου του array).

`ptr+i` και `&(ptr[i])` σημαίνουν το ίδιο πράγμα (διεύθυνση του i+1 στοιχείου του array).

Σημείωση: το μηδέν δεν είναι ποτέ έγκυρη διεύθυνση δεδομένων. ΄Οταν ένας δείκτης πάρει την τιμή 0 χρησιμοποιούμε και τη συμβολική σταθερά NULL. Το 0 χρησιμεύει για τη σηματοδότηση ανώμαλης κατάστασης. Εκτός από το 0 οι δείκτες και οι ακέραιοι δεν είναι εναλλάξιμοι, δηλαδή δε μπορούμε να χρησιμοποιούμε δείκτη στη θέση ακέραιου ή ακέραιο στη θέση δείκτη. Επίσης, δε μπορούμε να προσθέτουμε δείκτες. Η αριθμητική δεικτών είναι απλώς μια *συμβολική* αναφορά στη μετατόπιση κατά έναν αριθμό διευθύνσεων.

Με τη βοήθεια του τελεστή περιεχομένου και σε συνδυασμό με άλλους τελεστές όπως αυτός της αύξησης, μπορώ να εκφράσω με συνοπτικό τρόπο σύνθετες διαδικασίες. Παράδειγμα:

```
int *pa, *pb, i, a[10], b[10] = {0};

for (i=0; i<10, i++) a[i] = i;

pa = a;

pb = b;

for(i=0; i<10; i++) {

    *pb = *pa++;

    *pb++

}

for(i=0; i<a0, i++) printf("%d %d \n", a[i], b[i]);
```

Στο παραπάνω for όπου εξισώνονται τα στοιχεία του πίνακα, πρέπει να λάβουμε υπ' όψιν ότι η προτεραιότητα του τελεστή αύξησης είναι μεγαλύτερη από αυτή του τελεστή περιεχομένου. Έτσι, η εντολή

```
    *pb = *pa++;
```

αναλύεται ως εξής:

```
*pb = *pa;

pa = pa+1;
```

και μπορεί να "μεταφραστεί" από "γλώσσα" δεικτών σε "γλώσσα" πινάκων:

```
b[i] = a[i];

i = i+1;
```

Προσοχή! Μετά από τα παραπάνω, ο κάθε δείκτης δείχνει στη θέση μνήμης ακριβώς στο τελευταίο στοιχείο του αντίστοιχου πίνακα. Αν λοιπόν θέλαμε να τυπώσουμε τα στοιχεία χρησιμοποιώντας δείκτες αντί για στοιχεία πίνακα, θα έπρεπε να θυμηθούμε να επαναφέρουμε τους δείκτες στην αρχή, οπότε το πρόγραμμα θα ξαναγραφόταν έτσι:

.....

```
int *pa, *pb, i, a[10], b[10] = {0};

for (i=0; i<10, i++) a[i] = i;
```

```
pa = a;

pb = b;

for(i=0; i<10; i++) {

    *pb = *pa++;

    *pb++

}

pa = a;

pb = b;

for(i=0; i<a0, i++) printf("%d %d \n", *pa++, *pb++);
```

Δείκτες σε πίνακες χαρακτήρων.

Λόγω της ισοδυναμίας δεικτών-πινάκων, μπορούμε να ορίσουμε ένα αλφαριθμητικό ορίζοντας ένα δείκτη που να δείχνει στην αρχή του αλφαριθμητικού.

```
char *pstring = "Kozani city";

printf("%s\n",pstring);
```

Μάλιστα, αντίθετα από τους πίνακες, μπορώ να το κάνω να δείχνει αλλού:

```
pstring = "Ioannina city";
```

Αν ορίσω

```
char qstring[] = "Kozani city";
```

εδώ δε μπορώ να αλλάξω την τιμή του. Πάντως, μπορώ να τυπώσω:

```
printf("%s\n",qstring);
```

Αλλά, όπως και με τους πίνακες γενικότερα, η τιμή του είναι αμετάβλητη. Αν την πειράξω το πιθανότερο είναι να πάρω σφάλμα χρόνου εκτέλεσης.

Πίνακες δεικτών.

Οι πίνακες μπορούν να περιέχουν οποιοδήποτε είδος μεταβλητών αρκεί να είναι του ιδίου τύπου. Έτσι, μπορεί να περιέχουν και δείκτες. Αν οι δείκτες δείχνουν σε άλλους πίνακες, τότε μπορούμε να έχουμε μια πολύ ευέλικτη δομή δεδομένων γιατί μπορούμε να σχηματίσουμε κάτι σαν πίνακα που οι γραμμές του να έχουν διαφορετικό μήκος η κάθε μία. Για την ακρίβεια, κάθε στοιχείο του μπορεί να δείχνει σε πίνακα με διαφορετικό μήκος. Η πιο προφανής και συνήθης εφαρμογή είναι ένας πίνακας που αποθηκεύει αλφαριθμητικά διαφορετικού μήκους:

```
char *strings[4] = {"lala","lalala","lalalala","la"};
```

Παρατηρούμε ότι όπως και με τις απλές μεταβλητές, βάζουμε τον τελεστή * στη δήλωση.

Δεν είναι ούτε καν απαραίτητο οι πίνακες να είναι αποθηκευμένοι στη μνήμη με την ίδια σειρά που είναι δηλωμένοι στον πίνακα οι αντίστοιχοι δείκτες.

Τέλος, η ισοδυναμία δεικτών- πινάκων συνεπάγεται ότι ένας πίνακας με δείκτες μπορεί με τη σειρά του να αναπαρασταθεί από ένα δείκτη στο πρώτο του στοιχείο οδηγώντας **σε δείκτες σε**

δείκτες. Π.χ. σε σχέση με το παραπάνω μπορούμε να ορίσουμε

```
char **sp;

sp = strings;
```

Οι δείκτες παρέχουν ευελιξία και τη δυνατότητα πολύ συμπυκνωμένων εκφράσεων για σύνθετες διαδικασίες. Δεν πρέπει όμως να το παρακάνουμε γιατί υπάρχει κίνδυνος σύγχυσης και λαθών ενώ και το πρόγραμμα καταντά δυσνόητο.

Ορίσματα γραμμής εντολής: argc, *argv[] = αριθμός ορισμάτων και τιμές ορισμάτων (το 0 στοιχείο είναι το ίδιο το όνομα του προγράμματος)

Δυναμική διαχείριση μνήμης: malloc, calloc, realloc