

Κεφάλαιο

12

Πίνακες



Πίνακες (arrays)

Οι πίνακες αποτελούν ένα διαφορετικό τρόπο διαχείρισης της μνήμης. Ένας πίνακας είναι μια ομάδα θέσεων μνήμης στις οποίες γίνεται αναφορά με ένα κοινό όνομα. Μια συγκεκριμένη θέση μνήμης αυτής της ομάδας καθορίζεται από έναν ή περισσότερους αριθμούς αναφοράς. Στη C όλοι οι πίνακες αποτελούνται από συνεχόμενες θέσεις μνήμης και μπορεί να είναι μιας ή περισσότερων διαστάσεων.

Πίνακες μίας διάστασης

Έναν πίνακα μιας διάστασης τον φανταζόμαστε σαν μια ομάδα θέσεων μνήμης, τη μια πάνω από την άλλη. Κάθε θέση μνήμης έχει έναν αριθμό αναφοράς ο οποίος ξεκινάει από το 0 (η πρώτη 0, η δεύτερη 1 κ.ο.κ).

Στη C η πρόταση δήλωσης ενός πίνακα είναι παρόμοια με τις προτάσεις δήλωσης οποιασδήποτε άλλης μεταβλητής:

τύπος_δεδομένων όνομα_πίνακα[μέγεθος]; π.χ. η

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

δηλώνει ένα πίνακα με όνομα **a** και 10 θέσεις μνήμης.

Στη C η πρώτη θέση μνήμης ενός πίνακα έχει αριθμό αναφοράς 0. Επομένως, ένας πίνακας `int a[10]` θα έχει 10 θέσεις μνήμης από την `a[0]` μέχρι την `a[9]`.

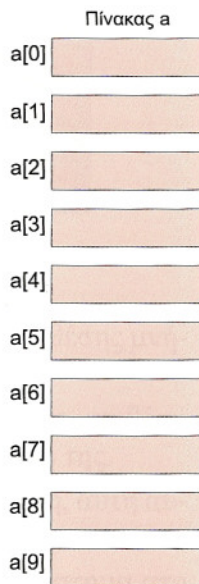
Όλες οι θέσεις μνήμης ενός πίνακα είναι του τύπου που δηλώθηκε στην πρόταση δήλωσης του πίνακα.

Οι θέσεις μνήμης ενός πίνακα χρησιμοποιούνται όπως οι υπόλοιπες μεταβλητές.

Η επόμενη πρόταση:

```
a[5]=45;
```

θα καταχωρίσει το 45 στην έκτη θέση μνήμης του πίνακα **a** (`a[0]` είναι η πρώτη θέση μνήμης, `a[5]` είναι η έκτη).



Συνήθως, για να προσδιορίσουμε μια θέση μνήμης ενός πίνακα χρησιμοποιούμε μια μεταβλητή. Αλλάζοντας την τιμή της μεταβλητής μπορούμε να έχουμε πρόσβαση σε διαφορετικές θέσεις του πίνακα.

Για παράδειγμα, το επόμενο τμήμα προγράμματος:

```
i=5;
a[i]=45;
i=8;
a[i]=64;
```

θα καταχωρίσει στη θέση `a[5]` του πίνακα `a` τον αριθμό 45 και στη θέση `a[8]` τον αριθμό 64, ενώ το επόμενο τμήμα προγράμματος:

```
for (i=0;i<=9;i++)
    a[i]=45;
```

θα γεμίσει όλες τις θέσεις μνήμης του πίνακα `a` με τον αριθμό 45.

Το παρακάτω πρόγραμμα γεμίζει τον πίνακα `a` με 10 τυχαίους αριθμούς και εμφανίζει στην οθόνη μόνον αυτούς που είναι μεγαλύτεροι από 1000:

```
main()
{
    int i,a[10];
    for(i=0;i<10;i++)
    {
        a[i]=rand();
    }
    for(i=0;i<10;i++)
    {
        if(a[i]>1000)
            printf("%d\n",a[i]);
    }
}
```


Οι πίνακες μίας διάστασης και οι δείκτες

Υπάρχει μια πολύ στενή σχέση μεταξύ πινάκων και δεικτών. Με απλά λόγια, **το όνομα ενός πίνακα είναι ένας δείκτης**.

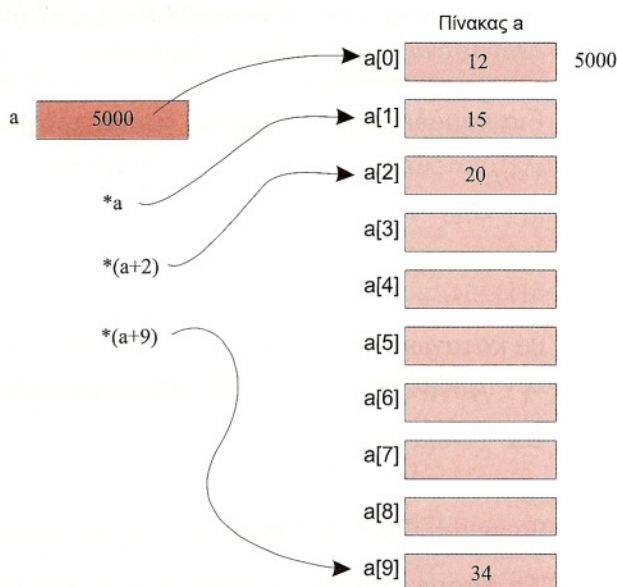
Ο δείκτης αυτός δείχνει σε θέσεις μνήμης του τύπου με τον οποίο δηλώθηκε ο πίνακας. Στο προηγούμενο παράδειγμα, με τη δημιουργία του πίνακα `a`, δημιουργείται στην ουσία και μια μεταβλητή δείκτη τύπου `int` με όνομα `a`.

Τις θέσεις μνήμης ενός πίνακα μπορούμε να τις προσπελάσουμε με δύο διαφορετικούς τρόπους: είτε χρησιμοποιώντας τον αριθμό αναφοράς τους μέσα στον πίνακα (`a[5]`), είτε χρησιμοποιώντας τη λογική των δεικτών:

`a[0]=12` ⇔ `*a=12`
`a[1]=15` ⇔ `*(a+1)=15`
`a[2]=20` ⇔ `*(a+2)=20`
`a[9]=34` ⇔ `*(a+9)=34`

Οι προηγούμενες παραστάσεις στα αριστερά είναι απολύτως ισοδύναμες με τις αντίστοιχες τους στα δεξιά.

Έτσι, η `*(a+9)` δείχνει στη θέση μνήμης με διεύθυνση 5036, δεδομένου ότι με την αριθμητική των δεικτών αν προστεθεί στο `a` (που έχει τιμή 5000) το 9 το αποτέλεσμα θα είναι 5036! Ο `a` είναι ένας δείκτης σε δεδομένα τύπου `int`, οπότε η πρόσθεση κάθε μονάδας στον `a` έχει αποτέλεσμα την αύξηση του κατά 4 (`a+9` ⇒ `5000 + 4*9` ⇒ `5036`).



Ποια είναι επομένως η διαφορά μεταξύ ενός πίνακα και ενός δείκτη;

Κάθε μία από τις παρακάτω δύο προτάσεις δημιουργούν ένα δείκτη `ptr`:

```
int ptr[100];
int *ptr;
```

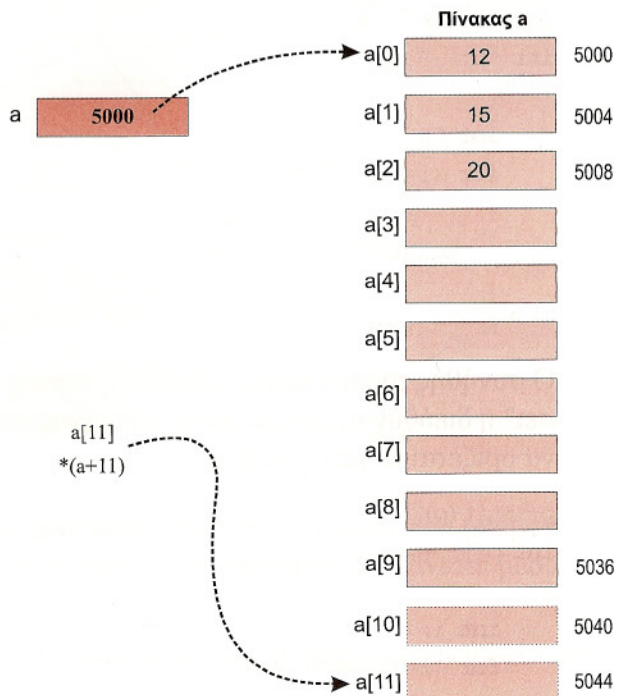
Το ερώτημα είναι, **είναι ισοδύναμες**;

Η απάντηση είναι **ΟΧΙ** επειδή η πρώτη πρόταση, εκτός του ότι δημιουργεί ένα δείκτη `ptr` δεσμεύει επίσης 100 θέσεις μνήμης τύπου `int` και **δίνει στο δείκτη αρχική τιμή** τη διεύθυνση της πρώτης θέσης μνήμης. Η δεύτερη πρόταση απλώς δημιουργεί ένα δείκτη χωρίς να του δίνει αρχική τιμή και χωρίς να δεσμεύει καμία άλλη θέση μνήμης.

Προσοχή Προσοχή !!!

Στις περισσότερες εκδόσεις της, η C **δεν** ελέγχει αν έχει ξεπεραστεί το όριο ενός πίνακα, με πολλές φορές (σχεδόν πάντα) καταστροφικά επακόλουθα.

Έτσι, στο προηγούμενο παράδειγμα ο πίνακας `a` έχει δέκα θέσεις, από την `a[0]` μέχρι την `a[9]`. Παρόλα αυτά μπορούμε κάλλιστα να προσπελάσουμε τη θέση `a[11]` χωρίς η C να δώσει κανένα μήνυμα λάθους. Το τι θα επακολουθήσει εξαρτάται μόνο από την τύχη μας.



Στη C ο έλεγχος των ορίων ενός πίνακα είναι στην ευθύνη του προγραμματιστή.

Μεταβίβαση ενός πίνακα ως παραμέτρου μιας συνάρτησης

Η μεταβίβαση παραμέτρων μιας συνάρτησης γίνεται πάντα με τη μεταβίβαση κατ' αξία (pass by value). Στην περίπτωση που η παράμετρος είναι ένας πίνακας, τότε μεταβιβάζουμε στη συνάρτηση τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα. Επομένως, η τυπική παράμετρος της συνάρτησης πρέπει να είναι ένας δείκτης του ίδιου τύπου όπως ο τύπος του πίνακα.

```
main()
{
    int array[100];
    ....
    printit(array);
}

printit(p)
int *p;
{
    int i;
    for (i=0; i<=99; i++)
    {
        printf("%d\n", p[i]);
    }
}
```

Η πρόσβαση στον πίνακα γίνεται μέσω του δείκτη p, σαν να ήταν κανονικός πίνακας. Ας μην ξεχνάμε ότι: $p[i] \Leftrightarrow *(p+i)$

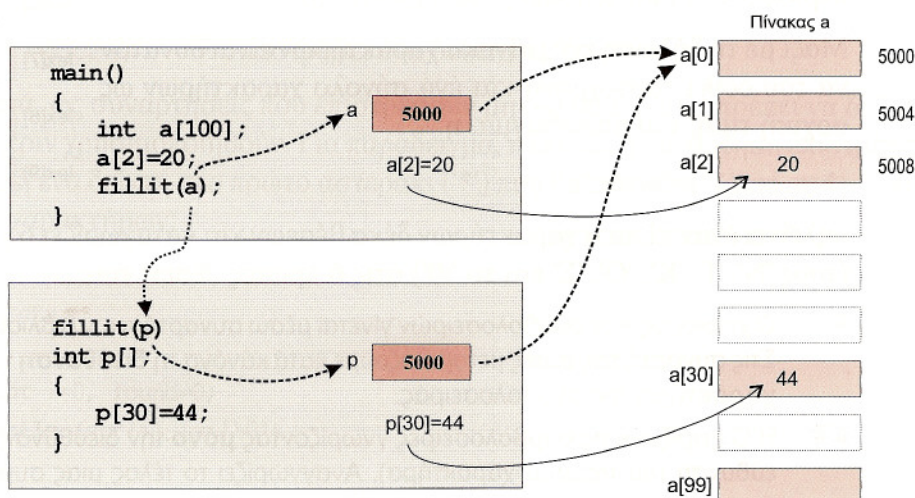
Ο συνήθης τρόπος που δηλώνεται η τυπική παράμετρος στην οποία θα "περάσει" η διεύθυνση ενός πίνακα, είναι ένας πίνακας του ίδιου τύπου αλλά χωρίς να ορίζεται το μέγεθος του:

```
printit(p)
int p[];
{
    int i;
    for (i=0; i<=99; i++)
    {
        printf("%d\n", p[i]);
    }
}
```

Η τυπική παράμετρος δηλώνεται σαν πίνακας χωρίς όμως διάσταση. Το αποτέλεσμα είναι η έμμεση δημιουργία ενός δείκτη p.

- ☞ Η τυπική παράμετρος της συνάρτησης στην οποία θα μεταβιβαστεί η διεύθυνση ενός πίνακα μιας διάστασης πρέπει να είναι ένας πίνακας (ή ένας δείκτης) ίδιου τύπου, χωρίς όμως να ορίζεται το μέγεθος του. Η δήλωση `p[]` δεν δημιουργεί νέο πίνακα στη συνάρτηση, ούτε δεσμεύει θέσεις μνήμης, αλλά δημιουργεί έμμεσα ένα δείκτη `p`.
- ☞ Μέσα στη συνάρτηση, η χρήση του πίνακα γίνεται κανονικά μέσω της τυπικής παραμέτρου. Για παράδειγμα, η `p[3]=12` θα βάλει το 12 στην τέταρτη θέση μνήμης του πίνακα `p`.

Στο επόμενο σχήμα-παράδειγμα ο πίνακας `a` δηλώνεται στη συνάρτηση `main()` και μεταβιβάζεται στη συνάρτηση `fillit()` μέσω της τυπικής της παραμέτρου `p`.



Μέσα από τη `main()` υπάρχει πρόσβαση στον πίνακα μέσω του ονόματος `a`, ενώ μέσα από τη `fillit()` η πρόσβαση **στον ίδιο πίνακα `a`** γίνεται μέσω του ονόματος `p`.

- ☞ Ο πίνακας `a` μέσα στη `fillit()` είναι άγνωστος. Πρόσβαση στον πίνακα `a` από την `fillit()` έχουμε μέσω του `p`.

Πίνακες χαρακτήρων



Η C δεν υποστηρίζει συγκεκριμένο τύπο μεταβλητής για την αποθήκευση συμβολοσειρών. **Οι αποθήκευση μιας συμβολοσειράς** (ή αλφαριθμητικού, *character string*) **γίνεται σε ένα μονοδιάστατο πίνακα χαρακτήρων**. Σε κάθε μία θέση του πίνακα αποθηκεύεται ένας χαρακτήρας της συμβολοσειράς.

Το σύνολο χαρακτήρων τερματίζεται πάντα με ένα χαρακτήρα null '\0'. Πρέπει λοιπόν ο πίνακας στον οποίο θα αποθηκευτεί μια συμβολοσειρά να έχει τουλάχιστον μία θέση περισσότερη από το πλήθος των χαρακτήρων.

Μαζί με τη δήλωση ενός πίνακα χαρακτήρων είναι δυνατόν να του δοθεί ταυτόχρονα και ένα σύνολο χαρακτήρων ως αρχική τιμή: Για παράδειγμα, η

```
char lex[10]="ΝΙΚΟΣ";
```

δηλώνει έναν πίνακα χαρακτήρων δέκα θέσεων και καταχωρίζει 6 χαρακτήρες (τους 'N', 'I', 'K', 'O', 'Σ' και το '\0') στις 6 πρώτες θέσεις του.

-  Ο χειρισμός των συμβολοσειρών γίνεται μέσω συναρτήσεων βιβλιοθήκης της C. Στις συναρτήσεις αυτές μεταβιβάζουμε κατά κανόνα τη **διεύθυνση του πρώτου χαρακτήρα** της συμβολοσειράς.
-  Η C χειρίζεται τις συμβολοσειρές γνωρίζοντας **μόνο** την διεύθυνσή τους (τη διεύθυνση του πρώτου χαρακτήρα). Αναγνωρίζει το τέλος μιας συμβολοσειράς από τον χαρακτήρα τερματισμού '\0'.

Οι συναρτήσεις που χρησιμοποιούνται για το χειρισμό των συμβολοσειρών δεν είναι παρά απλά προγράμματα χειρισμού των πινάκων στους οποίους είναι καταχωρισμένοι οι χαρακτήρες. Μπορούμε κάλλιστα να γράψουμε το δικό μας κώδικα και να μη χρησιμοποιήσουμε τις έτοιμες συναρτήσεις. Στη συνέχεια του κεφαλαίου αναφέρονται μερικές από τις συναρτήσεις χειρισμού των συμβολοσειρών. Στον ορισμό κάθε συνάρτησης αναφέρεται και το όνομα του αρχείου κεφαλίδας στο οποίο δηλώνεται. Αν θέλουμε να χρησιμοποιήσουμε κάποια από αυτές τις συναρτήσεις πρέπει στο πρόγραμμά μας να συμπεριλάβουμε με `#include` το αντίστοιχο αρχείο κεφαλίδας.

	char lex[10];
lex[0]	N
lex[1]	I
lex[2]	K
lex[3]	O
lex[4]	Σ
lex[5]	\0
lex[6]	
lex[7]	
lex[8]	
lex[9]	

Πριν όμως αναλύσουμε τις συναρτήσεις χειρισμού των συμβολοσειρών, ας εξετάσουμε έναν τύπο συνάρτησης που δεν έχουμε αναφέρει μέχρι τώρα: τις συναρτήσεις που επιστρέφουν ως τιμή ένα δείκτη.

Συναρτήσεις που επιστρέφουν ως τιμή ένα δείκτη

Μια συνάρτηση στη C μπορεί να επιστρέψει οποιαδήποτε τιμή· άρα, γιατί όχι και ένα δείκτη. Ουσιαστικά, μπορεί να επιστρέψει τη διεύθυνση μιας θέσης μνήμης. Στη δήλωση της συνάρτησης πρέπει να φαίνεται ότι η συνάρτηση επιστρέφει δείκτη καθώς και ο τύπος του δείκτη που επιστρέφεται. Οι επόμενες προτάσεις δηλώνουν συναρτήσεις που επιστρέφουν δείκτες:

char *func1 () ⇒ Επιστρέφει ένα δείκτη σε **char**

int *func2 () ⇒ Επιστρέφει ένα δείκτη σε **int**

Φυσικά, ως συναρτήσεις που επιστρέφουν μη αkéραιες τιμές πρέπει να δηλωθούν πριν χρησιμοποιηθούν. Για παράδειγμα, η επόμενη συνάρτηση επιστρέφει ως τιμή ένα δείκτη στο πρώτο αστεράκι (*) που θα εντοπίσει μέσα σε έναν πίνακα χαρακτήρων:

```
char *find(str)
char *str;
{
    int i=0, found=0;
    while (str[i] != '\0')
    {
        if (str[i] == '*')
        {
            found=1;
            break;
        }
        i++;
    }
    if (found) return(str+i); else return(NULL);
}
```

Στο επόμενο πρόγραμμα, η `find()` εντοπίζει το πρώτο * του πίνακα `lexi` και επιστρέφει ως τιμή ένα δείκτη που δείχνει στη θέση μνήμης στην οποία είναι το *. Έτσι, η `printf("%s\n", ptr)` εμφανίζει στην οθόνη το κομμάτι της φράσης από το * μέχρι το '\0'. Αν η `find()` δεν εντοπίσει * μέσα στη φράση, επιστρέφει ένα δείκτη NULL.

```
main()
{
    char lexi[40], *ptr;
    printf("Δώσε μία φράση");
    scanf("%s", lexi);
    ptr=find(lexi);
    if (ptr != NULL)
    {
        printf("Η φράση από το * και μετά είναι :");
        printf("%s\n", ptr);
    }
    else
        printf("Δεν υπάρχει * στη φράση που έδωσες");
}
```

Συναρτήσεις που εφαρμόζονται σε συμβολοσειρές

gets()

#include <stdio.h>

char *gets(char *str)

Η συνάρτηση `gets()` περιμένει να πληκτρολογηθούν χαρακτήρες και, μόλις πατηθεί το <ENTER>, τους αποθηκεύει έναν-έναν αρχίζοντας από τη θέση μνήμης στην οποία "δείχνει" ο δείκτης `str`.

Η συνάρτηση επιστρέφει ως τιμή τη διεύθυνση της πρώτης θέσης μνήμης από όπου άρχισε να αποθηκεύει τους χαρακτήρες. Ουσιαστικά, επιστρέφει την τιμή του δείκτη `str`.

- ☞ Ο δείκτης **str** πρέπει να είναι η διεύθυνση της πρώτης θέσης μνήμης ενός πίνακα τύπου **char**.
- ☞ Ο πίνακας πρέπει να έχει αρκετές θέσεις ώστε να αποθηκεύσει τους χαρακτήρες που θα πληκτρολογηθούν.
- ☞ Η **gets()** προσθέτει ένα χαρακτήρα null ('\0') στο τέλος του συνόλου χαρακτήρων. Πρέπει λοιπόν και αυτός ο χαρακτήρας να λαμβάνεται υπόψη για τον καθορισμό του μεγέθους του πίνακα.

```
main()
{
    char lexi[40];
    printf("Δώσε μία λέξη:");
    gets(lexi);
}
```

Το παραπάνω πρόγραμμα περιμένει να πληκτρολογηθούν χαρακτήρες από το χρήστη και τους αποθηκεύει στον πίνακα **lexi**. Αν πληκτρολογηθούν περισσότεροι από 40 χαρακτήρες, το πιθανότερο είναι ένα ωραίο "κρέμασμα" του προγράμματος.

puts()

```
#include <stdio.h>
```

```
int puts(char *str)
```

Η **puts()** γράφει ένα σύνολο χαρακτήρων στην οθόνη (την τυπική μονάδα εξόδου — *standard output*). Η παράμετρος είναι ένας δείκτης τύπου **char**, ο οποίος "δείχνει" σε κάποιο σύνολο χαρακτήρων (συμβολοσειρά). Στο τέλος η **puts()** προσθέτει και ένα χαρακτήρα αλλαγής γραμμής ('\n'). Η **puts()** επιστρέφει ως τιμή το 0 αν ήταν επιτυχής και μια μη μηδενική τιμή (όχι συγκεκριμένη) αν δεν είναι επιτυχής.


Η συνήθης χρήση της **puts()** είναι:

```
puts("C is the best");
```

που εμφανίζει στην οθόνη το συγκεκριμένο σύνολο χαρακτήρων ή

```
puts(lex);
```

που εμφανίζει στην οθόνη το σύνολο χαρακτήρων που βρίσκεται αποθηκευμένο μέσα στον πίνακα χαρακτήρων **lex**.

 Σε κάθε περίπτωση, στην **puts()** μεταβιβάζεται η διεύθυνση του πρώτου χαρακτήρα ενός συνόλου χαρακτήρων. Η **puts()** εμφανίζει έναν-έναν τους χαρακτήρες ξεκινώντας από την διεύθυνση που της δόθηκε μέχρι να εντοπίσει τον χαρακτήρα '\0', ο οποίος σηματοδοτεί το τέλος της συμβολοσειράς.

```
main()
{
    char lexi[40], *pp;
    pp="Η λέξη που έδωσες είναι:";
    puts("Δώσε μια λέξη :");
    gets(lexi);
    puts(pp);
    puts(lexi);
}
```

Δώσε μια λέξη:
C is fantastic
Η λέξη που έδωσες είναι:
C is fantastic

strlen()

```
#include <string.h>
```

```
int strlen(char *str)
```

Η **strlen()** επιστρέφει ως τιμή το πλήθος των χαρακτήρων της συμβολοσειράς στην οποία δείχνει ο δείκτης **str**.

```
main()
{
    char *pp, lexi[40];
    int len;
    pp="σκουληκομυρμηγκότρυπα";
    len=strlen(pp);
    printf("Η λέξη %s έχει %d χαρακτήρες\n",pp,len);
    printf("Δώσε μια λέξη :");
    gets(lexi);
}
```

```

len=strlen(lexi);
printf("Η λέξη που έδωσες έχει %d χαρακτήρες\n",len);
}

```

Το κομμάτι του κώδικα που υλοποιεί τη λειτουργία της `strlen()`, χρησιμοποιώντας αριθμητική δεικτών, είναι:

```

int mystrlen(str)
char *str;
{
    int count=0;
    while (*str != '\0')
    {
        count++;
        str++;
    }
    return(count);
}

```

και χρησιμοποιώντας πίνακες

```

int mystrlen(str)
char str[];
{
    int count=0;
    while (str[count] != '\0') count++;
    return(count);
}

```

strcmp()

```
#include <string.h>
```

```
int strcmp(char *str1, char *str2)
```


Η συνάρτηση `strcmp()` συγκρίνει αλφαβητικά δύο συμβολοσειρές και επιστρέφει ως αποτέλεσμα:

-1 ⇒ αν ο `str1` είναι μικρότερος από τον `str2`


0 ⇒ αν ο `str1` είναι ίσος με τον `str2`

1 \Rightarrow αν ο `str1` είναι μεγαλύτερος από τον `str2`

Οι παράμετροι `str1` και `str2` είναι δείκτες σε σύνολα χαρακτήρων.

 Πρέπει να γίνει κατανοητό ότι οι συμβολοσειρές συγκρίνονται αλφαβητικά και όχι αριθμητικά. Για παράδειγμα, `strcmp("ΝΙΚΟΣ", "ΑΛΕΞΑΝΔΡΟΣ")` έχει ως αποτέλεσμα 1 επειδή η συμβολοσειρά "ΝΙΚΟΣ" είναι **αλφαβητικά** μεγαλύτερη από τη συμβολοσειρά "ΑΛΕΞΑΝΔΡΟΣ".

Για να κατανοήσουμε τι σημαίνει αλφαβητικά μεγαλύτερο ή μικρότερο, φανταστείτε τις δύο συμβολοσειρές σαν καταχωρίσεις στον τηλεφωνικό κατάλογο. Αυτή που προηγείται είναι η μικρότερη.

 Για να είναι ίσες δύο συμβολοσειρές πρέπει να έχουν ακριβώς το ίδιο μήκος και να περιέχουν ακριβώς τους ίδιους χαρακτήρες. Για παράδειγμα, οι συμβολοσειρές "ΝΙΚΟΣ" και "ΝΙΚΟΣ " δεν είναι ίσες επειδή η δεύτερη περιέχει στο τέλος της δύο επιπλέον χαρακτήρες διαστήματος.

strcpy()

```
#include <string.h>
```


```
char *strcpy(char *str1, char *str2)
```

Η **strcpy()** αντιγράφει τη συμβολοσειρά που "δείχνει" ο δείκτης `str2` μέσα στο `str1`. Η συνάρτηση επιστρέφει ως τιμή ένα δείκτη στο `str1`.

Ο επόμενος κώδικας αντιγράφει τη λέξη "hello" στον πίνακα `str`.

```
char str[80];
```

```
strcpy(str, "hello");
```

 Η καταχώριση ενός συνόλου χαρακτήρων μέσα σε έναν πίνακα μπορεί να γίνει μόνο με την **strcpy()** ή με την **gets()**. Ένα σύνηθες λάθος που γίνεται είναι η επόμενη πρόταση καταχώρισης μιας συμβολοσειράς σε έναν πίνακα:

```
char lex[10];
```

```
lex="Νίκος";
```

Λάθος πρόταση καταχώρισης. Η καταχώριση χαρακτήρων μέσα σε πίνακα γίνεται με την `strcpy()` ή με την `gets()`.

Η παραπάνω πρόταση καταχώρισης **είναι λάθος**. Παρόλο που αρχικά φαίνεται να δουλεύει σωστά, είναι αιτία πολλών προβλημάτων που δύσκολα εντοπίζονται (βλέπε παράδειγμα Π5). Δεν πρέπει να γίνεται σύγχυση με τη σωστή καταχώριση αρχικής τιμής σε ένα πίνακα χαρακτήρων στην πρόταση δήλωσης του:

```
char lex[10]="Νίκος";
```

strcat()

```
#include <string.h>
```

```
char *strcat(char *str1, char *str2)
```

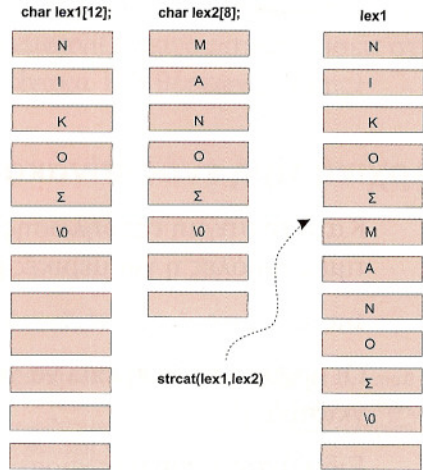
Η **strcat()** προθέτει στο τέλος του **str1** ένα αντίγραφο του **str2**.

Η **strcat()** επιστρέφει ως τιμή έναν δείκτη στην αρχή του συνόλου των χαρακτήρων. Ουσιαστικά, επιστρέφει τη διεύθυνση του **str1**.

Ο **str1** είναι ένας πίνακας στον οποίο έχει αποθηκευτεί ένα σύνολο χαρακτήρων. Ο πίνακας **str1** πρέπει να έχει αρκετές θέσεις ώστε να μπορεί να αποθηκεύσει και τους χαρακτήρες του **str2**.

Η επόμενη συνάρτηση προσθέτει τα περιεχόμενα του **lex2** στον **lex1**:

```
strcat(lex1,lex2);
```

**strstr()**

```
#include <string.h>
```

```
char *strstr(char *str1, char *str2)
```

Η **strstr()** αναζητάει τη συμβολοσειρά **str2** μέσα στη συμβολοσειρά **str1**. Η συνάρτηση επιστρέφει ως τιμή ένα δείκτη στη θέση όπου εντόπισε την **str2** μέσα στην **str1**. Σε περίπτωση που δεν εντοπιστεί η **str2** μέσα στην **str1**, επιστρέφει δείκτη null.

```
main()
```

```
{
```

```
    char frasi[200],lexi[20];
```

```
    printf("Δώσε μια φράση:");
```

```
    gets(frasi);
```

```
printf("Δώσε μια λέξη:");  
gets(lexi);  
if(strstr(frasi,lexi) != NULL)  
    printf("Η λέξη %s βρίσκεται μέσα στη φράση\n",lexi);  
else  
    printf("Η λέξη %s δεν βρίσκεται μέσα στη φράση\n",lexi);  
}
```

Αρχική τιμή ενός πίνακα μίας διάστασης

Κατά τη στιγμή της δήλωσης ενός πίνακα είναι δυνατόν να του δοθούν αρχικές τιμές για όλες ή για μερικές μόνο από τις θέσεις μνήμης του.

```
int a[10]={4,6,8,2,34,5,7,98,2,4};
```

Οι αρχικές τιμές πρέπει να χωρίζονται με κόμματα και να εσωκλείονται σε άγκιστρα {}.

Ειδικά στους πίνακες χαρακτήρων, η ανάθεση αρχικής τιμής γίνεται και με άλλον τρόπο:

```
char lex[20]="C is easy";
```

ο οποίος είναι ισοδύναμος με

```
char lex[20]={'C',' ','i','s',' ','e','a','s','y','\0'}
```

Σε κάθε μία θέση του πίνακα αποθηκεύονται με τη σειρά ένας-ένας οι χαρακτήρες. Πρέπει να προβλεφθεί θέση και για τον χαρακτήρα τερματισμού '\0' ο οποίος ακολουθεί κάθε σύνολο χαρακτήρων.

Στη C μπορούμε να δηλώσουμε έναν πίνακα χωρίς να καθορίσουμε το μέγεθός του αρκεί να δώσουμε στη δήλωσή του αρχικές τιμές:

```
char lex[]="C is easy";
```

Στην περίπτωση αυτή, θα δημιουργηθεί ένας πίνακας τόσων θέσεων όσων αρκούν για να χωρέσουν ίσα-ίσα οι αρχικές τιμές που δίνονται. Συγκεκριμένα για το παραπάνω παράδειγμα, θα δημιουργηθεί ένας πίνακας χαρακτήρων `lex` με 10 θέσεις (9 χαρακτήρες μαζί με τα διαστήματα και το '\0').

Χειρισμός πινάκων μίας διάστασης

Όταν θέλουμε να προσπελάσουμε όλες τις θέσεις μνήμης ενός πίνακα μιας διάστασης, ο καλύτερος τρόπος είναι μέσω ενός βρόχου `for`, με τη βοήθεια του οποίου θα αλλάζει ο αριθμός αναφοράς της θέσης μνήμης. Οι παρακάτω περιπτώσεις χρήσης ενός μονοδιάστατου πίνακα εμφανίζονται ως συναρτήσεις στις οποίες μεταβιβάζουμε ως παράμετρο έναν πίνακα 100 θέσεων μνήμης.

Συμπλήρωση πίνακα με τυχαίες τιμές:

```
fill_it(a)
int a[];
{
    int i;
    for (i=0;i<100;i++)
    {
        a[i]=rand();
    }
}
```

Η πρόταση αυτή εκτελείται 100 φορές με τιμές του `i` από 0 μέχρι 99. Κάθε φορά που εκτελείται, ένας τυχαίος αριθμός καταχωρίζεται σε μια θέση μνήμης του πίνακα: την πρώτη φορά στην `a[0]`, τη δεύτερη στην `a[1]`, κ.ο.κ

Εμφάνιση περιεχομένων πίνακα:

```
print_it(a)
int a[];
{
    int i;
    for (i=0;i<100;i++)
    {
        printf("%d\n",a[i]);
    }
}
```

Η πρόταση αυτή εκτελείται 100 φορές με τιμές του `i` από 0 μέχρι 99. Κάθε φορά που εκτελείται, εμφανίζονται στην οθόνη τα περιεχόμενα μιας θέσης μνήμης του πίνακα: την πρώτη φορά της `a[0]`, τη δεύτερη της `a[1]`, κ.ο.κ

Υπολογισμός αθροίσματος περιεχομένων πίνακα:


```
athroisma(a)
int a[];
{
    int i, sum;
    sum=0;
    for (i=0; i<100; i++)
    {
        sum=sum+a[i];
    }
    printf("Το άθροισμα των στοιχείων του πίνακα είναι: %d\n", sum);
}
```

Κάθε φορά που εκτελείται αυτή η πρόταση, προστίθενται στη *sum* τα περιεχόμενα μιας θέσης μνήμης του πίνακα. Την πρώτη φορά της *a[0]*, τη δεύτερη της *a[1]*, κ.ο.κ. Στο τέλος η *sum* θα περιέχει το άθροισμα όλων των θέσεων μνήμης του πίνακα.

Υπολογισμός ελάχιστης και μέγιστης τιμής περιεχομένων πίνακα:

```
minmax(a)
int a[];
{
    int min, max, i;
    min=max=a[0];
    for (i=0; i<100; i++)
    {
        if(a[i]<min) min=a[i];
        if(a[i]>max) max=a[i];
    }
    printf("Ο μικρότερος αριθμός είναι: %d\n", min);
    printf("Ο μεγαλύτερος αριθμός είναι: %d\n", max);
}
```

Κάθε φορά που εκτελούνται αυτές οι προτάσεις, ελέγχεται η τιμή μιας θέσης μνήμης του πίνακα αν είναι μεγαλύτερη από τη μέχρι στιγμής μεγαλύτερη ή μικρότερη από τη μέχρι στιγμής μικρότερη τιμή που έχει βρεθεί. Αν όντως βρεθεί μικρότερη ή μεγαλύτερη τιμή, τότε αυτή αντικαθιστά την υπάρχουσα τιμή στις μεταβλητές *min* και *max* αντίστοιχα. Αυτό γίνεται για κάθε θέση μνήμης και τελικά η *min* και η *max* θα περιέχουν τη μικρότερη και τη μεγαλύτερη τιμή αντίστοιχα από όλα τα στοιχεία του πίνακα.

 Οι μεταβλητές **min** και **max** "περιέχουν" τη μέχρι στιγμής μικρότερη και μεγαλύτερη τιμή που έχει βρεθεί. **ΠΡΟΣΟΧΗ** στην αρχική τους τιμή. Πρέπει να είναι μια υπάρχουσα τιμή από τα στοιχεία του πίνακα. Συνήθως αναθέτουμε και στις δύο μεταβλητές ως αρχική τιμή το περιεχόμενο της πρώτης θέσης μνήμης του πίνακα.

Εύρεση μιας τιμής μέσα σε πίνακα:

```

find_it(a)
int a[];
{
    int found, ar, i;
    printf("Τιμή για εύρεση:");
    scanf("%d", &ar);
    found=0;
    for (i=0; i<100; i++)
    {
        if(a[i]==ar) found=1;
    }
    if (found==1)
        printf("Ο αριθμός βρέθηκε");
    else
        printf("Ο αριθμός δεν βρέθηκε");
}

```

Κάθε φορά που εκτελείται αυτή η πρόταση, ελέγχεται η τιμή μιας θέσης μνήμης του πίνακα αν είναι ίση με τον αριθμό που δόθηκε για εύρεση. Στην περίπτωση που βρεθεί ίση τιμή, τότε καταχωρίζεται στη θέση found το 1.

Μετά το τέλος του βρόχου for ελέγχεται η τιμή της found. Αν είναι 1, αυτό σημαίνει ότι ο ζητούμενος αριθμός βρέθηκε, διαφορετικά όχι. **ΠΡΟΣΟΧΗ** στην αρχική τιμή της found. Πρέπει να είναι 0.

Πίνακες πολλών διαστάσεων

Η C υποστηρίζει πίνακες με περισσότερες από μία διάσταση. Η δήλωση ενός πίνακα με πολλές διαστάσεις γίνεται κατανοητή με το επόμενο παράδειγμα

```
int pin[4][6][10];
```

όπου δηλώνεται ένας πίνακας τριών διαστάσεων με όρια,

0~3 ⇨ για την πρώτη διάσταση

0~5 ⇨ για τη δεύτερη διάσταση

0~9 ⇨ για την τρίτη διάσταση

Ο πίνακας έχει συνολικά $4 \cdot 6 \cdot 10 = 240$ θέσεις μνήμης

Όταν θέλουμε να προσπελάσουμε όλες τις θέσεις μνήμης ενός πίνακα πολλών διαστάσεων, ο ενδεδειγμένος τρόπος είναι η χρήση ένθετων βρόχων **for**, με τη βοήθεια των οποίων αλλάζουν οι αριθμοί αναφοράς για κάθε θέση μνήμης. Για

παράδειγμα, το επόμενο πρόγραμμα γεμίζει έναν πίνακα τριών διαστάσεων με τυχαίους αριθμούς:

```
main()
{
    int pin[4][6][10], i, j, k;
    for (i=0; i<4; i++)
        for (j=0; j<6; j++)
            for (k=0; k<10; k++)
                pin[i][j][k]=rand();
}
```

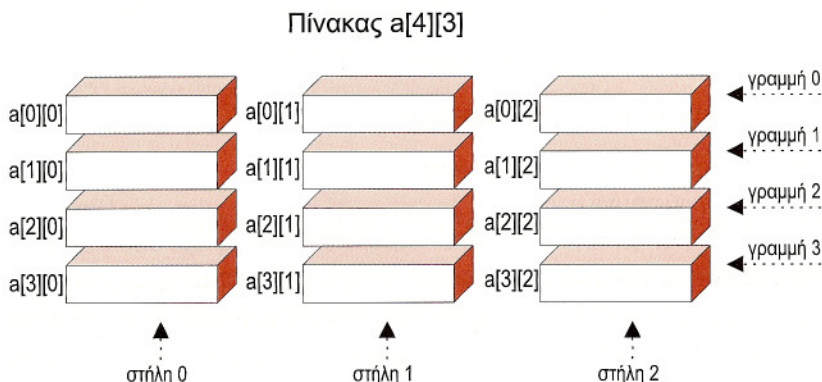
Πίνακες δύο διαστάσεων

Ιδιαίτερη αναφορά θα κάνουμε στους πίνακες δύο διαστάσεων, οι οποίοι είναι αυτοί που χρησιμοποιούνται πιο συχνά από τους πολυδιάστατους πίνακες.

Φανταζόμαστε έναν πίνακα δύο διαστάσεων σαν μια ομάδα θέσεων μνήμης διατεταγμένων σε σειρές και στήλες. Κάθε θέση μνήμης έχει δύο αριθμούς αναφοράς, από τους οποίους ο ένας προσδιορίζει τη γραμμή και ο άλλος τη στήλη στην οποία βρίσκεται η θέση μνήμης.

Ας θεωρήσουμε τον παρακάτω πίνακα **a**, 12 θέσεων, με 4 γραμμές και 3 στήλες που ορίζεται από τη δηλωτική πρόταση:

```
int a[4][3];
```



Η προσπέλαση στις θέσεις μνήμης γίνεται με τη χρήση και των δυο διαστάσεων:

`a[1][2]` ⇒ Προσδιορίζει τη θέση που βρίσκεται στη γραμμή 1 (δεύτερη γραμμή) και στη στήλη 2 (τρίτη στήλη).

Όταν θέλουμε να προσπελάσουμε όλες τις θέσεις μνήμης ενός πίνακα δύο διαστάσεων, χρησιμοποιούμε δύο ένθετους βρόχους `for`, μέσω των οποίων αλλάζουν οι αριθμοί αναφοράς για κάθε διάσταση. Ο ένας βρόχος αλλάζει την τιμή του αριθμού αναφοράς για την πρώτη διάσταση και ο άλλος για τη δεύτερη.

Ο επόμενος κώδικας γεμίζει έναν πίνακα τεσσάρων γραμμών και τριών στηλών με τυχαίους αριθμούς:

```
main()
{
    int a[4][3], i, j;
    for (i=0; i<4; i++)
        for (j=0; j<3; j++)
            a[i][j]=rand();
}
```

Πίνακες δύο διαστάσεων και δείκτες

Όπως γίνεται με τους πίνακες μιας διάστασης, μαζί με τη δήλωση ενός πίνακα δύο διαστάσεων, η C δημιουργεί και ένα δείκτη (pointer) με όνομα ίδιο με το όνομα του πίνακα και αρχική τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα.

Για να έχει πρόσβαση σε μια θέση μνήμης ενός πίνακα η C, πρέπει με κάποιον τρόπο να υπολογίσει τη διεύθυνση της συγκεκριμένης θέσης μνήμης. Για τον υπολογισμό αυτό, ο εσωτερικός μηχανισμός της γλώσσας χρησιμοποιεί τη φιλοσοφία και την αριθμητική των δεικτών.

Στους πίνακες μιας διάστασης ο υπολογισμός αυτός είναι σχετικά εύκολος. Έτσι, η διεύθυνση της θέσης μνήμης `a[5]` προκύπτει αν προσθέσουμε στην αρχική διεύθυνση του πίνακα `a` (π.χ. το 5000) το 5, χρησιμοποιώντας πάντα την

αριθμητική των δεικτών. Αν ο πίνακας είναι τύπου `char` το αποτέλεσμα θα είναι 5005, ενώ αν είναι τύπου `int` θα είναι 5020 ($5000+5*4$).

Πώς γίνεται ο υπολογισμός της διεύθυνσης σε έναν πίνακα δύο διαστάσεων;

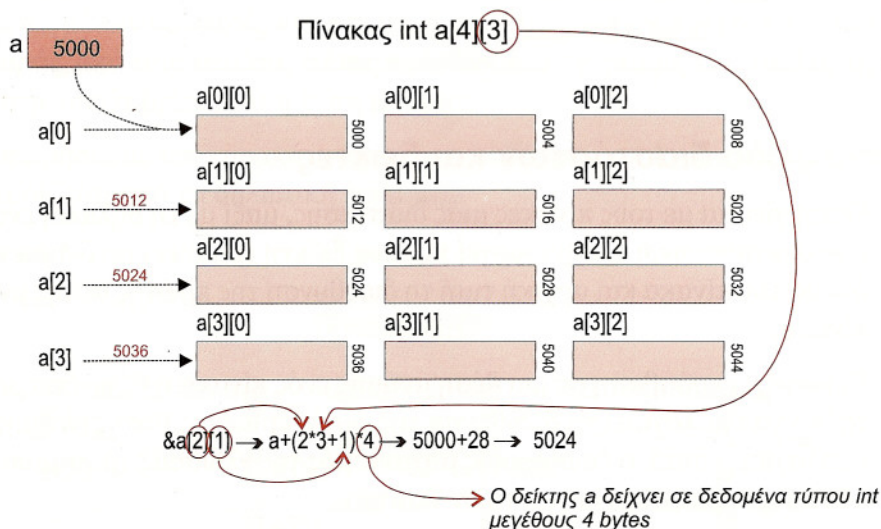
Ας θεωρήσουμε τη γενική δήλωση ενός πίνακα `a` δύο διαστάσεων:

τύπος `a[διάσταση1][διάσταση2]`

Η διεύθυνση μιας θέσης μνήμης `a[x][y]` του πίνακα υπολογίζεται από την παράσταση: $a + (x * \text{διάσταση2} + y) * \text{byte}$

όπου `a` το περιεχόμενο του δείκτη `a`, δηλαδή η διεύθυνση της πρώτης θέσης μνήμης του πίνακα `a`, `byte` το μέγεθος του τύπου του πίνακα (`char` 1, `int` 4, `float` 4, `double` 8), και `διάσταση2` είναι η μέγιστη τιμή της δεύτερης διάστασης όπως ορίζεται στη δηλωτική πρόταση του πίνακα.

Για παράδειγμα, με τη δήλωση `int a[4][3]` δηλώνεται ο εξής πίνακας `a`:



Εσωτερικά, για να χρησιμοποιήσει η C τη θέση π.χ. `a[2][1]` πρέπει πρώτα να υπολογίσει τη διεύθυνσή της, δηλαδή το 5028. Για να υπολογιστεί η διεύθυνση της θέσης `a[2][1]` του παραπάνω πίνακα, στην παράσταση $a + (x * \text{διάσταση2}$

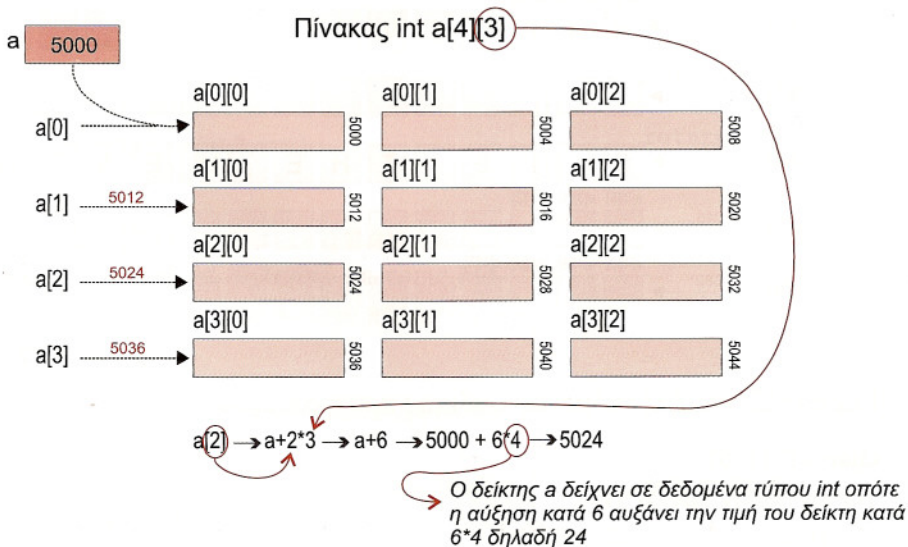
+ y)*byte, αντικαθίσταται το **a** με 5000, το **x** με 2, το **διάσταση2** με 3, το **y** με 1, το **byte** με 4 (ο πίνακας είναι τύπου `int`) και προκύπτει το εξής αποτέλεσμα: $5000 + (2*3+1)*4 = 5028$ που είναι η διεύθυνση της θέσης `a[2][1]`.

Με ανάλογο τρόπο υπολογίζεται η διεύθυνση μιας θέσης μνήμης και σε πίνακες περισσότερων διαστάσεων.

☞ Η αναφορά στον τρόπο υπολογισμού της διεύθυνσης μιας θέσης μνήμης ενός πίνακα γίνεται για την πλήρη κατανόηση των διεργασιών που ακολουθεί η C στον χειρισμό των πινάκων. Δεν μας είναι άμεσα χρήσιμη, δεδομένου ότι ποτέ δεν θα χρειαστεί να την υπολογίσουμε εμείς.

☞ Παρατηρούμε ότι για τον υπολογισμό της διεύθυνσης μιας θέσης μνήμης (βλέπτε τον τρόπο υπολογισμού στο προηγούμενο σχήμα) χρησιμοποιείται η μέγιστη τιμή μόνο της δεύτερης διάστασης του πίνακα. Αυτό ας το σημειώσουμε γιατί με βάση αυτό, θα εξηγήσουμε αργότερα τον τρόπο μεταβίβασης ενός πίνακα πολλών διαστάσεων σε συνάρτηση.

Πέρα από τη δημιουργία ενός δείκτη που δείχνει στην πρώτη θέση μνήμης του πίνακα, χρησιμοποιώντας μόνο την πρώτη διάσταση μπορούμε να έχουμε δείκτες στις διαφορετικές γραμμές ενός πίνακα δύο διαστάσεων:

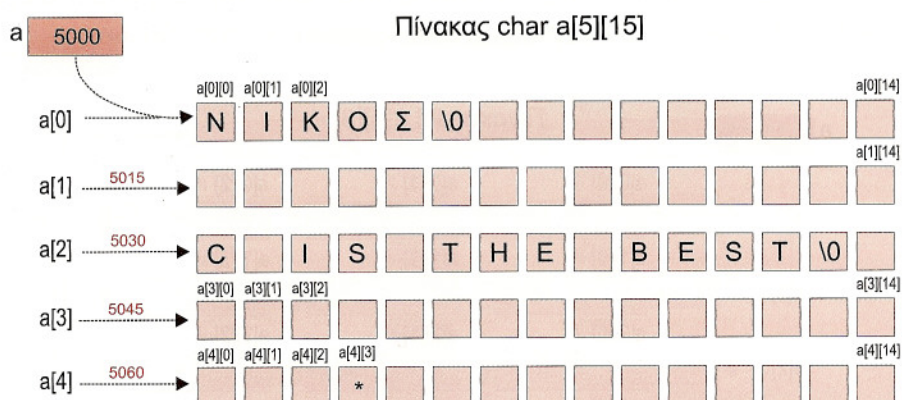


Η αναφορά `a[0]` επιστρέφει τη διεύθυνση της πρώτης θέσης της πρώτης γραμμής του πίνακα, η `a[1]` επιστρέφει τη διεύθυνση της πρώτης θέσης της δεύτερης γραμμής του πίνακα, η `a[2]` επιστρέφει τη διεύθυνση της πρώτης θέσης της τρίτης γραμμής του πίνακα κ.ο.κ. Στο προηγούμενο σχήμα φαίνεται ο τρόπος υπολογισμού της διεύθυνσης που επιστρέφει η αναφορά `a[2]`.

👉 Η παράσταση `a[2]` είναι ισοδύναμη με την `&a[2][0]`.

Πίνακες δύο διαστάσεων για αποθήκευση σειρών χαρακτήρων

Πίνακες χαρακτήρων δύο διαστάσεων χρησιμοποιούνται για αποθήκευση συμβολοσειρών. Σε κάθε γραμμή του πίνακα καταχωρίζεται ένα σύνολο χαρακτήρων. Η γραμμή πρέπει να έχει ικανό αριθμό θέσεων ώστε να μπορεί να χωρέσει τους χαρακτήρες συμπεριλαμβανομένου και του χαρακτήρα τερματισμού `\0` που ακολουθεί κάθε συμβολοσειρά.



Για παράδειγμα, ο επόμενος κώδικας προγράμματος

```
char a[5][15];
strcpy(a[0], "NIKOΣ");
strcpy(a[2], "C IS THE BEST");
a[4][3] = '*';
```


καταχωρίζει στην πρώτη και τρίτη γραμμή του πίνακα τα αντίστοιχα σύνολα χαρακτήρων. Στην τελευταία γραμμή του πίνακα στη θέση `a[4][3]` καταχωρίζεται ένας αστερίσκος (*). Μια γραμμή του πίνακα `a` θα μπορούσε να αποθηκεύσει και ένα σύνολο χαρακτήρων που θα πληκτρολογηθεί από το πληκτρολόγιο, π.χ. η

```
gets(a[4]);
```

περιμένει να πληκτρολογηθούν από το πληκτρολόγιο χαρακτήρες και μόλις πληκτρολογηθεί το <ENTER> τους αποθηκεύει στην πέμπτη γραμμή του πίνακα `a`. Αν πληκτρολογηθούν περισσότεροι από δεκατέσσερις χαρακτήρες (δεκαπέντε μαζί με το \0), τότε τα αποτελέσματα θα είναι απρόβλεπτα.

Μεταβίβαση πινάκων πολλών διαστάσεων σε συναρτήσεις

Όπως αναφέραμε προηγουμένως, κατά τη μεταβίβαση ενός πίνακα μιας διάστασης ως παραμέτρου μιας συνάρτησης, αρκεί η μεταβίβαση του ονόματος του πίνακα, δηλαδή ενός δείκτη, στην πρώτη θέση μνήμης του πίνακα.

```
main()
{
    int a[10], sum;
    sum=findsum(a);
    printf("sum=%d\n", sum);
}
```

Στη συνάρτηση `findsum()` η τυπική παράμετρος `x` μπορεί να δηλωθεί με τρεις διαφορετικούς τρόπους, με το ίδιο πάντα αποτέλεσμα:

- `int *x;` ⇨ σαν ένας δείκτης σε δεδομένα τύπου `int`
- `int x[];` ⇨ σαν ένας πίνακας χωρίς διαστάσεις
- `int x[10];` ⇨ σαν ένας πίνακας με συγκεκριμένες διαστάσεις.

```
findsum(x)
int *x;
{
    int ss=0;
    for (i=0;i<10;i++)
        ss=ss+x[i];
    return(ss);
}
```

```
findsum(x)
int x[];
{
    int ss=0;
    for (i=0;i<10;i++)
        ss=ss+x[i];
    return(ss);
}
```

Και στις τρεις περιπτώσεις, το αποτέλεσμα είναι η δημιουργία ενός δείκτη **x** μέσα στον οποίο θα αντιγραφεί η διεύθυνση της πρώτης θέσης μνήμης του **a**.

- ✎ Αξιοσημείωτο είναι ότι στον τρίτο τρόπο δήλωσης της τυπικής παραμέτρου **x** (**int x[10]**) η C **δεν** δεσμεύει 10 θέσεις όπως στην κανονική δήλωση ενός πίνακα. Στην ουσία, το 10 το αγνοεί.
- ✎ Ο συνήθης τρόπος δήλωσης της τυπικής παραμέτρου για πίνακα μιας διάστασης είναι ο δεύτερος, με άλλα λόγια η δήλωση της τυπικής παραμέτρου ως πίνακα χωρίς διαστάσεις.

Στην περίπτωση που έχουμε πίνακες περισσότερων διαστάσεων, τότε η συνάρτηση πρέπει να γνωρίζει εκτός από τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα και τις μέγιστες τιμές των διαστάσεών του (εκτός της πρώτης) για να μπορεί να υπολογίσει τη διεύθυνση μιας θέσης μνήμης του πίνακα.

Αν υποθέσουμε ότι έχουμε έναν πίνακα **a** δύο διαστάσεων 10x10, τον οποίο θέλουμε να μεταβιβάσουμε ως παράμετρο σε μια συνάρτηση:

```
main()
{
    int a[10][10];
    .....
    func(a);
}
func(x)
int x[][10];
{
    .....
}
```

Στη δήλωση της παραμέτρου **x** πρέπει να δηλωθεί η τιμή της δεύτερης διάστασης του πίνακα. Στην περίπτωση αυτή, η **x** δεν μπορεί να δηλωθεί απλώς ως δείκτης.

Στη περίπτωση πινάκων με περισσότερες από δύο διαστάσεις, στη δήλωση της τυπικής παραμέτρου στην οποία θα μεταβιβαστεί η διεύθυνση της πρώτης θέσης μνήμης του πίνακα, δηλώνονται όλες οι τιμές των διαστάσεών του εκτός από την πρώτη. Μπορεί να δηλωθεί και η τιμή της πρώτης διάστασης αλλά δεν είναι απαραίτητο:

```

main()
{
    int a[10][10][7][8];
    .....
    func(a);
}

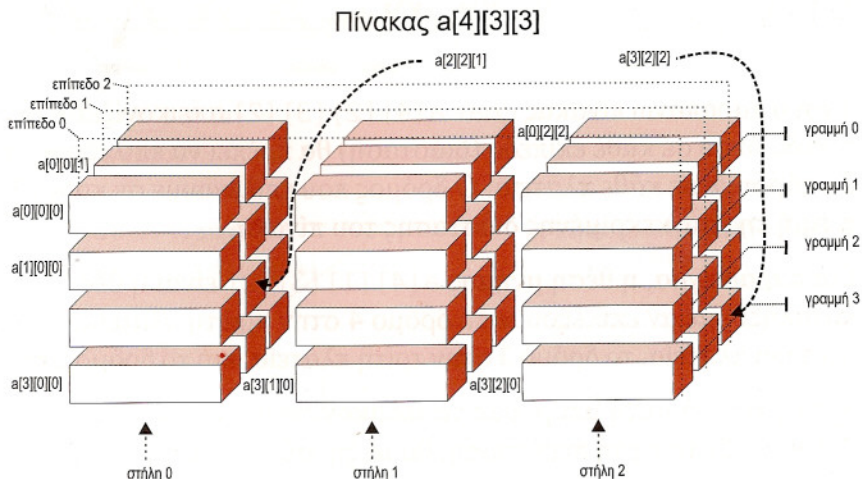
func(int x[][10][7][8])
{
    .....
}

```

Απεικόνιση πινάκων πολλών διαστάσεων

Αν και στην πραγματικότητα, οι θέσεις ενός πίνακα πολλών διαστάσεων, είναι συνεχόμενες, αρκετές φορές είναι πολύ βολικό να έχουμε στο μυαλό μας μια εποπτική και σίγουρα πιο παραστατική εικόνα ενός πίνακα. Έναν πίνακα τριών διαστάσεων τον φανταζόμαστε σαν θέσεις μνήμης διατεταγμένες σε γραμμές, στήλες, αλλά και επίπεδα.

Στο επόμενο παράδειγμα, ο πίνακας **a** τριών διαστάσεων (**a[4][3][3]**) απεικονίζεται σαν θέσεις μνήμης σε διάταξη τεσσάρων γραμμών, τριών στηλών, και τριών επιπέδων.

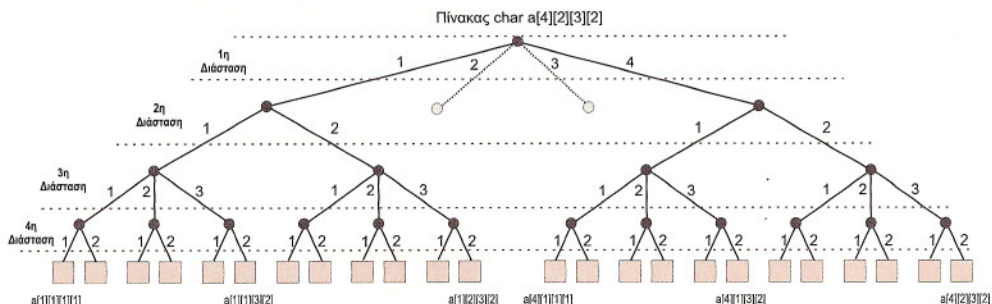


Για να αναφερθούμε σε μια θέση μνήμης πρέπει να καθορίσουμε τη γραμμή, τη στήλη, και το επίπεδο στο οποίο βρίσκεται.

Χρησιμοποιώντας τις τρεις διαστάσεις του χώρου, μπορούμε να απεικονίσουμε πίνακες μέχρι τριών διαστάσεων. Πώς μπορούμε όμως να φανταστούμε εποπτικά έναν πίνακα περισσότερων διαστάσεων; Είναι δυνατόν να απεικονίσουμε στο χαρτί έναν πίνακα τεσσάρων, πέντε ή και περισσότερων διαστάσεων;

Η απάντηση είναι ΝΑΙ αλλά θα πρέπει να ξεχάσουμε τον τρισδιάστατο χώρο που χρησιμοποιούσαμε μέχρι στιγμής και να σκεφτούμε τις θέσεις μνήμης σαν χωριά στα οποία για να φτάσουμε θα πρέπει να επιλέξουμε το σωστό δρόμο σε κάθε σταυροδρόμι.

Έτσι, έναν πίνακα τεσσάρων διαστάσεων τον φανταζόμαστε σαν μια συστοιχία δρόμων οι οποίοι ξεκινάνε από μία πλατεία. Κάθε δρόμος καταλήγει σε μια άλλη πλατεία κ.ο.κ Τελικά για να φτάσουμε σε ένα χωριό (θέση μνήμης) θα πρέπει να επιλέξουμε ποιο δρόμο θα πάρουμε σε κάθε μία από τις τέσσερις πλατείες που περνάμε με τη σειρά.



Έτσι ο παραπάνω πίνακας `char a[4][2][3][2]` απεικονίζεται με τέσσερα επίπεδα όπου σε κάθε επίπεδο (διάσταση) θα πρέπει να επιλέξουμε ποιο δρόμο θα πάρουμε σε κάθε πλατεία. Ο δρόμος που επιλέγουμε σε κάθε πλατεία είναι η τιμή της συγκεκριμένης διάστασης του πίνακα.

Για παράδειγμα, η θέση μνήμης `a[4][1][3][2]` είναι η θέση στην οποία θα καταλήξουμε αν επιλέξουμε το δρόμο 4 στην πρώτη πλατεία, το δρόμο 1 στη δεύτερη πλατεία, το δρόμο 3 στην τρίτη πλατεία, και το δρόμο 2 στην τέταρτη.

Με τη λογική αυτή μπορούμε να απεικονίσουμε πίνακες με όσες διαστάσεις θέλουμε. Σε αυτή τη περίπτωση, θα αυξηθούν τα επίπεδα και για να καταλή-

ξουμε σε κάποια θέση μνήμης, θα πρέπει να περάσουμε από περισσότερες "πλατείες" — τόσες όσες οι διαστάσεις του πίνακα.

Παραδείγματα

- Π.1** Η επόμενη συνάρτηση αντιστρέφει τα περιεχόμενα ενός πίνακα ακεραίων 100 θέσεων. Με άλλα λόγια, η πρώτη θέση γίνεται τελευταία, η δεύτερη προτελευταία, κ.ο.κ.

```
void reverse(a)
int a[];
{
    int i,temp;
    for(i=0;i<50;i++)
    {
        temp=a[99-i];
        a[99-i]=a[i];
        a[i]=temp;
    }
}
```

Αντιμεταθέτουμε τα περιεχόμενα των θέσεων i και $99-i$. Δηλαδή της 0 με την 99, της 1 με την 98 κ.λπ. Αυτό όμως πρέπει να γίνει για i από 0 μέχρι 49, διαφορετικά (αν γίνει μέχρι το 99) ο πίνακας θα αντιστραφεί ξανά και θα είναι όπως στην αρχή.

- Π.2** Η επόμενη συνάρτηση υπολογίζει και εμφανίζει τα αθροίσματα των γραμμών ενός πίνακα ακεραίων δύο διαστάσεων 10 γραμμών και 10 στηλών (10 x 10).

```
void print_sum(a)
int a[][10];
{
    int i,j,sum;
    for(i=0;i<10;i++)
    {
        sum=0;
        for(j=0;j<10;j++)
        {
            sum=sum+a[i][j];
        }
    }
}
```

Στη θέση `sum` καταχωρίζεται το 0 πριν από κάθε νέα γραμμή.

Στη `sum` αθροίζονται όλες οι θέσεις της γραμμής i .

```

    }
    printf("Αθροισμα γραμμής %d = %d\n", i, sum);
}
}

```

Π.3 Το επόμενο πρόγραμμα ζητάει ένα σύνολο χαρακτήρων, διαγράφει τα κενά διαστήματα που ενδεχομένως περιέχει και το εμφανίζει στην οθόνη. Για παράδειγμα, αν πληκτρολογήσουμε "Η C είναι ωραία" εμφανίζει "HCείναιωραία".

```

main()
{
    char lex[40];
    int i=0,j,c=0;
    printf("Δώσε μια φράση:");
    gets(lex);
    while (lex[i]!='\0')
    {
        if(lex[i]==' ')
        {
            j=i;
            while(lex[j]!='\0')
            {
                lex[j]=lex[j+1];
                j++;
            }
            c++;
            i--;
        }
        i++;
    }
    puts(lex);
    printf("Αφαίρεσα %d κενά\n",c);
}

```

Η φράση καταχωρίζεται στον πίνακα lex.

Ελέγχονται ένας-ένας οι χαρακτήρες μέχρι να φτάσουμε στο τέλος των χαρακτήρων.

Ελέγχει αν ο χαρακτήρας είναι το διάστημα.

Μετακινεί όλους τους χαρακτήρες από το διάστημα και κάτω μια θέση πάνω.

Η μεταβλητή c μετράει τα κενά που βρέθηκαν.

Π.4 Η επόμενη συνάρτηση αναζητάει μία λέξη μέσα σε έναν πίνακα χαρακτήρων δύο διαστάσεων. Ο πίνακας έχει 100 γραμμές και κάθε γραμμή του πίνακα περιέχει μία φράση μήκους μέχρι 50 χαρακτήρες. Ο πίνακας καθώς και η λέξη που αναζητάμε μεταβιβάζονται ως παράμετροι στη συνάρτηση. Αν η συνάρτηση εντοπίσει τη λέξη, επιστρέφει ένα δείκτη στη γραμμή του πίνακα στην οποία βρήκε τη λέξη, διαφορετικά επιστρέφει δείκτη NULL.

```
char *find_lexi(char lista[][50], char lexi[])
{
    int i;
    for(i=0;i<100;i++)
    {
        if(strstr(lista[i],lexi)!=NULL)
            return lista[i];
    }
    return NULL;
}
```

Η συνάρτηση χρησιμοποιείται ως εξής:

```
main()
{
    char basi_lexeon[100][50];
    char lexi_anaz[20];
    char *p;
    .....
    printf("Δώσε λέξη για αναζήτηση:");
    gets(lexi_anaz);
    p=find_lexi(basi_lexeon,lexi_anaz);
    if(p!= NULL)
        printf("Βρέθηκε η λέξη μέσα στη φράση %s\n",p);
    else
        printf("Δεν βρέθηκε η λέξη μέσα στον πίνακα\n");
}
```

Π.5 Όπως αναφέρθηκε στη σελίδα 224, ένα σύνηθες λάθος που γίνεται είναι οι χρήση των επόμενων προτάσεων για τη καταχώριση μιας συμβολοσειράς σε έναν πίνακα:

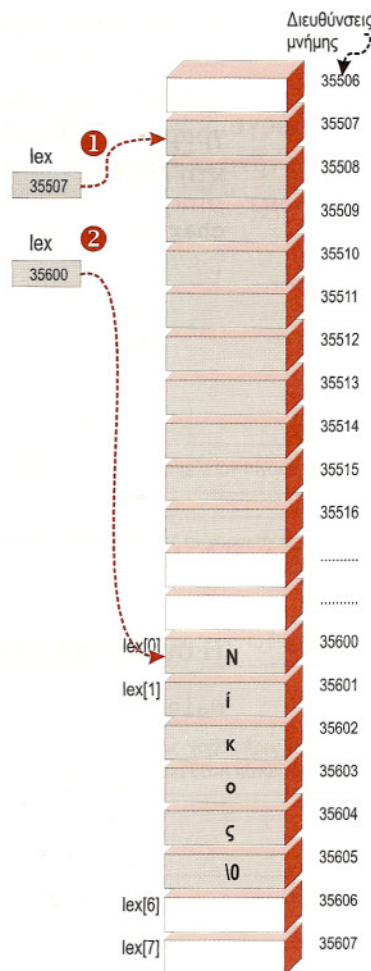
❶ `char lex[10];`

❷ `lex="Νίκος";`

Ο τρόπος αυτός αρχικά φαίνεται να δουλεύει σωστά είναι όμως εντελώς λάθος. Ας εξηγήσουμε το γιατί:

Στο στάδιο της μεταγλώττισης, η C δεσμεύει τις 10 θέσεις του πίνακα `lex` και ταυτόχρονα δημιουργεί έναν δείκτη `lex` με αρχική τιμή τη διεύθυνση της πρώτης από τις θέσεις που δεσμεύτηκαν. Στο διπλανό σχήμα υποθέτουμε ότι δεσμεύτηκαν οι θέσεις (35507~35516) και ο δείκτης `lex` περιέχει τη διεύθυνση 35507. Επίσης υποθέτουμε ότι η συμβολοσειρά "Νίκος", της πρότασης 2, αποθηκεύεται στη μνήμη ξεκινώντας από τη διεύθυνση 35600 η οποία αποτελεί τη τιμή της συμβολοσειράς.

Η πρόταση 2, έχει σαν αποτέλεσμα, η τιμή της συμβολοσειράς "Νίκος" (το 35600) να καταχωριστεί στον δείκτη `lex`. Οπότε από το σημείο αυτό και μετά όταν αναφερόμαστε στον πίνακα `lex` αναφερόμαστε στις θέσεις μνήμης που ξεκινάνε από τη διεύθυνση 35600 και όχι στις 10 δεσμευμένες θέσεις 35507~35516. Π.χ η θέση μνήμης `lex[1]` αναφέρεται στη διεύθυνση 35601, η `lex[6]` στη διεύθυνση 35606 κ.ο.κ. Εμείς θεωρώντας ότι ο πίνακας `lex` έχει 10 θέσεις μπορεί να προσπαθήσουμε να τις χρησιμοποιήσουμε. Οι θέσεις μνήμης όμως μετά από την 35605 δεν έχουν δεσμευτεί και πιθανώς να χρησιμοποιούνται από άλλο τμήμα κώδικα. Η χρήση αυτών των θέσεων πιθανότατα θα οδηγήσει σε "κρέμασμα" του προγράμματος.



Ανασκόπηση Κεφαλαίου 12

- Ένας πίνακας είναι μια διάταξη θέσεων μνήμης υπό ένα κοινό όνομα.
- Οι πίνακες που χρησιμοποιούνται πιο συχνά είναι οι πίνακες μιας και δύο διαστάσεων. Μπορούμε όμως να ορίσουμε πίνακες περισσότερων διαστάσεων ανάλογα με τις ανάγκες μας.
- Ένας πίνακας ορίζει έμμεσα ένα δείκτη με αρχική τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα.
- Η δήλωση ενός πίνακα γίνεται όπως και οποιασδήποτε άλλης μεταβλητής και ακολουθείται από το πλήθος και την τιμή των διαστάσεων του. Π.χ. η `int a[4][5]` δηλώνει έναν πίνακα `a`, δύο διαστάσεων, με είκοσι θέσεις (4x5), με την πρώτη διάσταση από το 0 μέχρι το 3 και τη δεύτερη από το 0 μέχρι το 4.
- Η αναφορά σε μια θέση ενός πίνακα γίνεται με το όνομά του και τους αριθμούς αναφοράς για την κάθε διάσταση, ώστε να προσδιοριστεί η θέση μνήμης μέσα στον πίνακα. Για παράδειγμα, η `a[2][3]` αναφέρεται στη θέση μνήμης με αριθμούς αναφοράς 2 και 3 για την πρώτη και δεύτερη διάσταση αντίστοιχα.
- Το όνομα ενός πίνακα αποτελεί ταυτόχρονα και ένα δείκτη με τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα. Ο χειρισμός ενός πίνακα μπορεί να γίνει και με τη χρήση δεικτών.
- Για την αποθήκευση ενός συνόλου χαρακτήρων απαιτείται ένας πίνακας χαρακτήρων. Για παράδειγμα, ο πίνακας `char lex[30]` επιτρέπει την καταχώριση ενός συνόλου το πολύ 30 χαρακτήρων (μαζί με το χαρακτήρα τερματισμού `'\0'`).
- Ένας πίνακας χαρακτήρων δύο διαστάσεων χρησιμοποιείται για την καταχώριση ενός αριθμού από σύνολα χαρακτήρων.
- Ο χειρισμός των συνόλων χαρακτήρων γίνεται με τις συναρτήσεις βιβλιοθήκης `strcpy()`, `strcmp()`, `strlen()`, `strcat()`, `gets()`, `puts()`, κ.λπ.
- Ο χειρισμός των πινάκων γίνεται συνήθως με τη χρήση βρόχων `for`. Για να έχουμε πρόσβαση σε όλες τις θέσεις μνήμης ενός πίνακα, ανάλογα με το πλήθος των διαστάσεών του, χρησιμοποιούνται και αντίστοιχοι ένθετοι βρόχοι `for`.

- Μια συνάρτηση μπορεί να επιστρέφει ως τιμή ένα δείκτη. Μια τέτοια συνάρτηση δηλώνεται, όπως και οι υπόλοιπες, με τον τελεστή * πριν από το όνομά της: `char *func()` δηλώνει ότι η συνάρτηση `func()` επιστρέφει ως τιμή ένα δείκτη σε `char`.
- Όταν μεταβιβάζουμε έναν πίνακα ως παράμετρο σε μια συνάρτηση, στην ουσία μεταβιβάζουμε τη διεύθυνση της πρώτης θέσης μνήμης του. Η τυπική παράμετρος πρέπει να είναι ένας πίνακας του ίδιου τύπου, όπου υποχρεωτικά πρέπει να δηλώσουμε τη μέγιστη τιμή των διαστάσεων του πλην της πρώτης. Για παράδειγμα, `a[]` για τη μεταβίβαση ενός πίνακα μιας διάστασης, `a[][4][5]` για τη μεταβίβαση ενός πίνακα τριών διαστάσεων.

Ασκήσεις Κεφαλαίου 12

- 12.1** Έστω δύο πίνακες `lexi1` και `lexi2` μέσα στους οποίους αποθηκεύονται με την `gets()` δύο λέξεις: ★★ ★

```
main()
{
    char lexi1[80], lexi2[80];
    gets(lexi1);
    gets(lexi2);
}
```

- Να γραφεί συνάρτηση η οποία να εμφανίζει στην οθόνη τα κοινά γράμματα των δύο λέξεων από μία φορά το καθένα.
- Να γραφεί συνάρτηση η οποία να διαγράφει από τον `lexi1` όσους χαρακτήρες περιέχονται στον `lexi2`.
- Να γραφεί συνάρτηση η οποία να βρίσκει αν η λέξη `lexi2` υπάρχει μέσα στη `lexi1`. Να επιστρέφει 0 αν δεν υπάρχει και, αν υπάρχει, να επιστρέφει τον αριθμό της θέσης μνήμης του `lexi1` από την οποία αρχίζει η `lexi2`.

- 12.2** Να γραφεί συνάρτηση με όνομα `convert()` η οποία να μετατρέπει τους λατινικούς χαρακτήρες ενός πίνακα χαρακτήρων σε πεζούς ή κεφαλαίους. Η συνάρτηση να ορίζεται όπως παρακάτω: ★ ★

```
int convert(char *str, int sel)
```

Η παράμετρος `str` είναι ένας δείκτης σε `char` και δείχνει στον πίνακα χαρακτήρων που θα μετατραπεί. Η `sel` καθορίζει τη λειτουργία της συνάρτησης ως εξής:

- Αν η `sel` έχει τιμή 1, η συνάρτηση θα μετατρέψει τα πεζά σε κεφαλαία.
- Αν η `sel` έχει τιμή 0, η συνάρτηση θα μετατρέψει τα κεφαλαία σε πεζά.
- Η συνάρτηση θα επιστρέφει ως τιμή τον αριθμό των χαρακτήρων που μετατράπηκαν.

Να λάβετε υπόψη ότι οι λατινικοί κεφαλαίοι και πεζοί χαρακτήρες διαφέρουν κατά 32. Για παράδειγμα, ο κωδικός του 'A' είναι 65 και του 'a' 97.

- 12.3** Τι κάνει η επόμενη συνάρτηση; Τι τιμή επιστρέφει (με λόγια); ★ ★ ★

```
char *blablabla(char *str1, char *str2, int num)
{
    int i=0;
    while((str1[i] != '\0') && (i<num))
    {
        str2[i]=str1[i];
        i++;
    }
    str2[num]='\0';
    return str2;
}
```

- 12.4** Τι θα γινόταν αν καλούσαμε τη συνάρτηση από το επόμενο πρόγραμμα και δίναμε τη λέξη "Παπατρεχαγυρευόπουλος"; ★ ★

```
main()
{
    char lex1[40], lex2[40];
    puts("Δωσε μια λέξη");
    gets(lex1);
    blablabla(lex1,lex2,7);
    puts(lex2);
}
```

12.5 Δεδομένου του πίνακα **a** όπως φαίνεται στο διπλανό σχήμα, τι αποτελέσματα θα έχει το επόμενο πρόγραμμα; ★ ★

```
void func2();
void func3();

main()
{
    char a[10], *ptr;
    .....
    func1(a[5]);
    func2(a);
    func2(&a[5]);
    func2(a+5);
    func3(a,5);
    func3(a+5,2);
}

int func1(p)
char p;
{
    return p+1;
}
```

Πίνακας a

a[0]	B
a[1]	E
a[2]	N
a[3]	E
a[4]	T
a[5]	I
a[6]	A
a[7]	\0
a[8]	
a[9]	


```
void func2(p)
char *p;
{
    puts(p);
}
```

```
void func3(p,num)
char *p;
int num;
{
    int i;
    for(i=num; i>=0; i--)
        putchar(p[num]);
}
```

12.6 Δεδομένου του πίνακα **a** όπως φαίνεται στο διπλανό σχήμα, τι αποτελέσματα θα έχει το επόμενο πρόγραμμα; ★★

```
char *func6();

main()
{
    char a[10], *ptr;
    .....
    printf("RES1=%d\n", func5(a, 'E'));
    printf("RES2=%d\n", func5(a+5, 'E'));
    ptr=func6(a, 'I');
    if(ptr!=NULL) printf("To %c υπάρχει στον a\n", *ptr);
}

int func5(char *p, char ch)
{
    int i=0, cnt=0;
    while(p[i] != '\0')
    {
```

Πίνακας a

a[0]	B
a[1]	E
a[2]	N
a[3]	E
a[4]	T
a[5]	I
a[6]	A
a[7]	\0
a[8]	
a[9]	

```

        if(p[i] == ch) cnt++;
        i++;
    }
    return cnt;
}

char *func6(char *p, char ch)
{
    int i=0;
    while (p[i] != '\0')
    {
        if (p[i] == ch) return &p[i];
        i++;
    }
    return NULL;
}

```

12.7 Δεδομένου του πίνακα **a** όπως φαίνεται στο διπλανό σχήμα, τι αποτελέσματα θα έχει στον πίνακα η κλήση της επόμενης συνάρτησης: ★★

```

func4(a,a+6);
.....
void func4(p1,p2)
char *p1,*p2;
{
    char ch;
    while(p1<p2)
    {
        ch=*p1;
        *p1=*p2;
        *p2=ch;
        p1++;
        p2--;
    }
}

```

Πίνακας a

a[0]	B
a[1]	E
a[2]	N
a[3]	E
a[4]	T
a[5]	I
a[6]	A
a[7]	\0
a[8]	
a[9]	

12.8 Τι κάνει η επόμενη συνάρτηση: ★

```
func1(pin)
int pin[][10];
{
    int i,j;
    for (i=0;i<35;i++)
        for (j=0;j<10;j++)
            pin[i][j]=i*j;
}
```

- Αν καλέσουμε τη συνάρτηση με όρισμα έναν πίνακα με όνομα `test`: `func1(test)`, πόσων και τι διαστάσεων πρέπει να είναι ο `test` για να δουλέψει σωστά η συνάρτηση `func()`;
- Ποιό το περιεχόμενο της θέσης `pin[30][5]`;
- Αν η δήλωση `int pin[][10]` ήταν `int pin[5][10]` θα δούλευε σωστά η συνάρτηση;

12.9 Να γραφεί συνάρτηση η οποία να δέχεται ως παράμετρο έναν πίνακα `int` τριών διαστάσεων 10x20x5 και να επιστρέφει ως τιμή το άθροισμα όλων των θέσεων μνήμης του πίνακα. ★

12.10 Να γραφεί πρόγραμμα το οποίο να ζητάει 100 αριθμούς, να τους καταχωρίζει σε έναν πίνακα μιας διάστασης, και να υπολογίζει το μέσο όρο τους. ★

12.11 Να γραφεί πρόγραμμα το οποίο να καταχωρίζει σε έναν τετραγωνικό πίνακα (με ίδιο αριθμό γραμμών και στηλών) τον ίδιο αριθμό στα κελιά της διαγωνίου του από τα ΒΔ προς τα ΝΑ. ★

8					
	8				
		8			
			8		
				8	
					8

- 12.12** Να γραφεί πρόγραμμα το οποίο να καταχωρίζει σε έναν τετραγωνικό πίνακα (με ίδιο αριθμό γραμμών και στηλών) έναν αριθμό στα κελιά της διαγωνίου του από ΒΔ προς ΝΑ, έναν άλλον αριθμό στο δεξιό του τμήμα και έναν τρίτο αριθμό στο αριστερό του. Να χρησιμοποιηθεί ένας μόνο βρόχος. ★★

1	8	8	8	8	8
5	1	8	8	8	8
5	5	1	8	8	8
5	5	5	1	8	8
5	5	5	5	1	8
5	5	5	5	5	1

- 12.13** Τι αποτέλεσμα θα έχει στον διπλανό πίνακα **a** ο επόμενος κώδικας: ★★

```
main()
{
    int a[6][6];
    int i,j;
    for(i=0;i<6;i++)
    {
        for(j=0;j<6;j++)
        {
            if(j+i>5)
                a[i][j]=1;
            else if(j+i<5)
                a[i][j]=2;
            else
                a[i][j]=0;
        }
    }
}
```


- 12.14** Να γραφεί συνάρτηση η οποία θα επιστρέφει ως τιμή το άθροισμα των στοιχείων της διαγωνίου ενός τετραγωνικού πίνακα `int a[100][100]`. ★★

12.15 Με δεδομένο έναν πίνακα `int a[100][20]` να γραφεί κώδικας ο οποίος να εμφανίζει τον μεγαλύτερο αριθμό από κάθε σειρά του πίνακα (θα εμφανίζει 100 αριθμούς). ★ ★ ★

12.16 Ποια από τα επόμενα αληθεύουν: ★

- ☐ Ένας πίνακας ορίζει έμμεσα και ένα δείκτη με αρχική τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα.
- ☐ Σε ένα πίνακα χαρακτήρων μιας διάστασης μπορούν να καταχωριστούν πολλές συμβολοσειρές.
- ☐ Για να έχει μια συνάρτηση πρόσβαση σε έναν πίνακα μιας διάστασης, πρέπει να της διαβιβάσουμε μόνο τη διεύθυνση του πίνακα.
- ☐ Για να έχει μια συνάρτηση, πρόσβαση σε έναν πίνακα περισσότερων από μιας διαστάσεων, πρέπει γνωρίζει εκτός από τη διεύθυνση του πίνακα και τις τιμές των διαστάσεών του εκτός από την πρώτη.
- ☐ Κατά τη δήλωση ενός πίνακα, η τιμές των διαστάσεών του μπορούν να είναι και μεταβλητές. Π.χ. η `int a[b]` δημιουργεί έναν πίνακα `a` με θέσεις μνήμης όσες και η τιμή της μεταβλητής `b`.