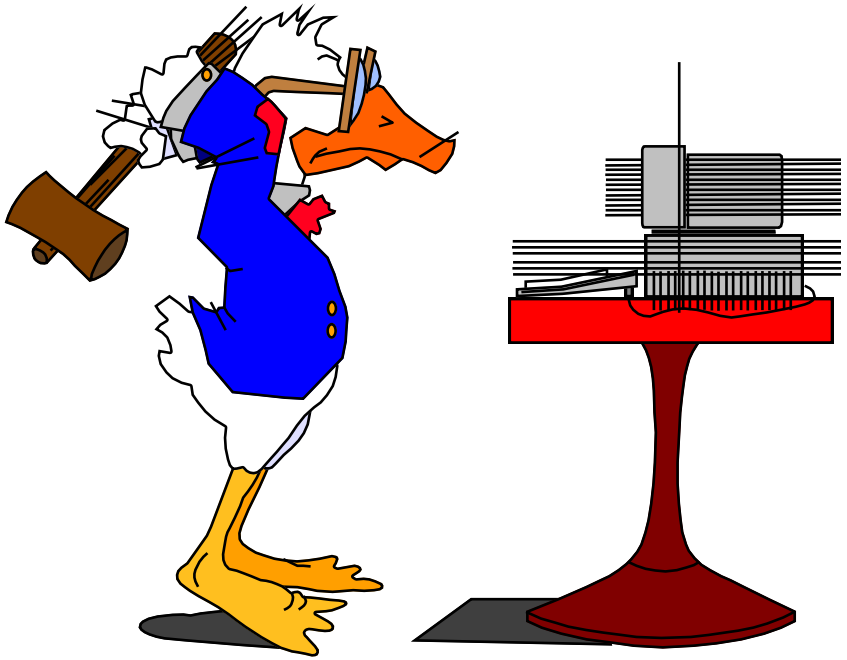
Οι δείκτες στη γλώσσα C

Δείκτης είναι μία μεταβλητή η οποία περιέχει σαν τιμή μία διεύθυνση της μνήμης

Η τιμή ενός δείκτη δείχνει σε μία άλλη μεταβλητή, η οποία μπορεί να προσεγγισθεί έμμεσα με τους ειδικούς τελεστές δεικτών

*** και &**

ΟΙ ΔΕΙΚΤΕΣ ΣΤΗ C



Η σωστή χρήση των δεικτών θεωρείται πολύ σημαντική επιτυχία για κάθε προγραμματιστή της γλώσσας C .

Οι δείκτες στη γλώσσα C

Ο τελεστής * προσδιορίζει το περιεχόμενο μιας μεταβλητής της οποίας η διεύθυνση είναι η τιμή ενός δείκτη.

Μπορούμε να θυμόμαστε τον τελεστή * περιφραστικά με την έκφραση "στη διεύθυνση".

Ο τελεστής & επιστρέφει τη διεύθυνση μιας μεταβλητής και μπορούμε να τον θυμόμαστε περιφραστικά με την έκφραση "η διεύθυνσή της".

Π.χ. αν a και b είναι 2 ακέραιες μεταβλητές και h είναι ένας ακέραιος δείκτης τότε οι εντολές :

```
h = &a;
b = *h;
```

μεταβιβάζουν την τιμή της μεταβλητής a στη μεταβλητή b.

Οι Δείκτες σαν διευθύνσεις

memory address (hex)	contents							
0000	1	1	0	0	0	1	1	0
0001	1	0	1	1	1	0	0	0
0002	0	1	1	1	1	1	0	0
....								
....								
....								
00FF	1	1	1	1	0	0	0	1
....								
....								
....								
FF00	1	0	0	1	1	0	0	0
....								
....								
....								
FFFF	1	1	1	0	1	1	1	1

a και b ακέραιες μεταβλητές στις διευθύνσεις 00FF και FF00
h ακέραιος δείκτης

h=&a;
b=*h;

Ο δείκτης παίρνει την τιμή OOFF και δίνει το περιεχόμενο αυτής της διεύθυνσης στη μεταβλητή b

Παράδειγμα

```c
#include <stdio.h>

int main()
{
    int *h;
    int a=10,b=20;
    h = &a;
    b = *h;
    printf("Η τιμή του a είναι %d\n", *h);
    printf("Η τιμή του *h είναι %d\n", *h);
    printf("Η τιμή του b είναι %d\n", b);
    printf("Η τιμή του h είναι %p\n", h);
    printf("Η διεύθυνση του a είναι %p\n", &a);
    return(0);
}
```

Έξοδος

```
[Running] cd "/Users/eleftheriakatsiri/Downloads/lab_codes/" && gcc p3.c -o p3 && "/Users/
eleftheriakatsiri/Downloads/lab_codes/"p3
Η τιμή του a είναι 10
Η τιμή του *h είναι 10
Η τιμή του b είναι 10
Η τιμή του h είναι 0x7ff7b46ec58c
Η διεύθυνση του a είναι 0x7ff7b46ec58c

[Done] exited with code=0 in 1.449 seconds
```

Σαν το ποδήλατο …

Κουράγιο:

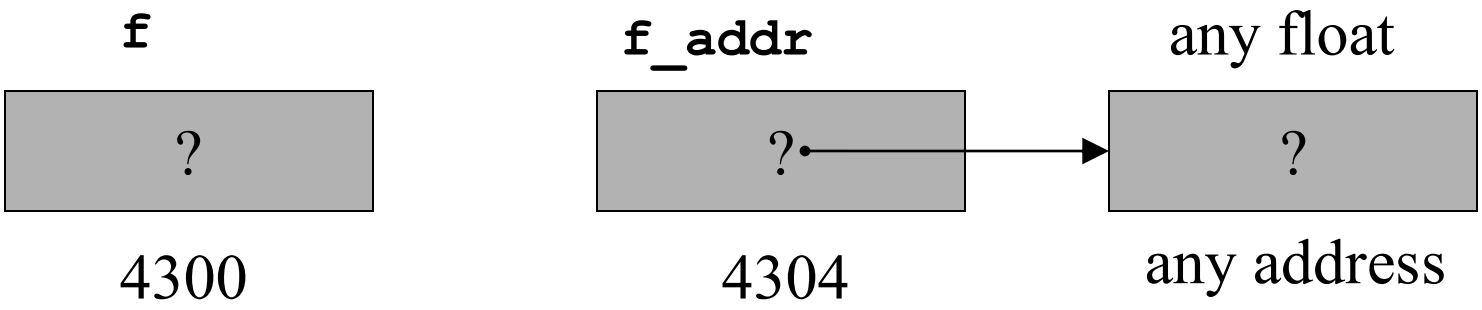
Η εκμάθηση της χρήσης δεικτών είναι σαν να μαθαίνει κανείς ποδήλατο: εκεί που θεωρεί ότι είναι αδύνατο να μάθει τους δείκτες, ξαφνικά τα καταφέρνει! Επιπλέον, εάν μάθει κανείς να χρησιμοποιεί δείκτες, δύσκολα θα χάσει αυτή τη δεξιότητα.

πριν

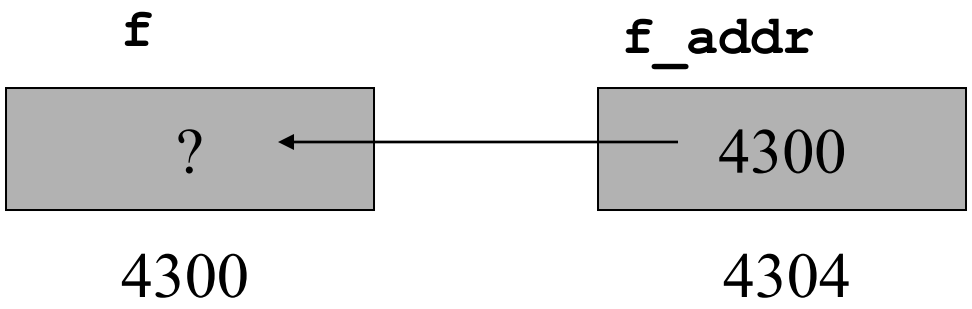
μετά

Δείκτες-Pointers (1)

```
float f;          /* απλή μεταβλητή */
float *f_addr;    /* μεταβλητή δείκτη */
```

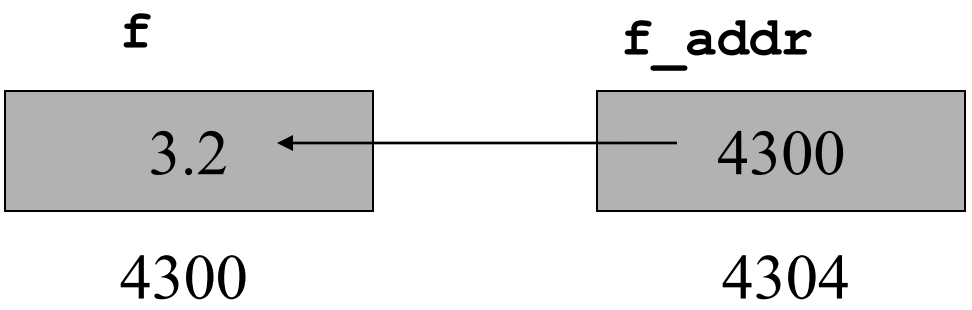



```
f_addr = &f;     /* & = τελεστής διεύθυνσης */
```

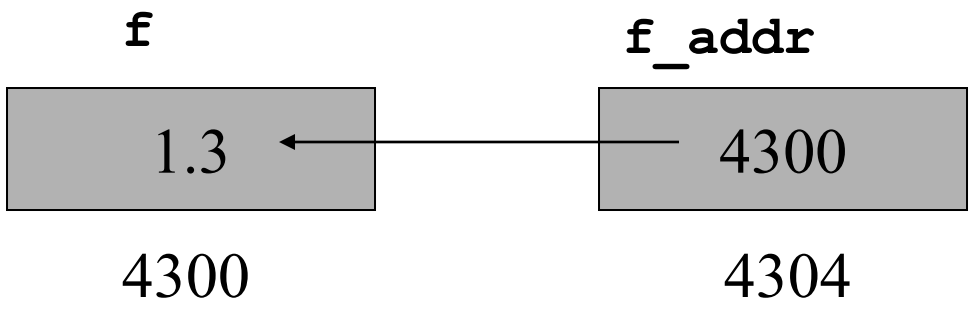


Δείκτες-Pointers (2)

```
*f_addr = 3.2; /* * = τελεστής έμμεσης διεύθυνσης */
```




```
float g = *f_addr;/* Η τιμή του g γίνεται 3.2 */
      f = 1.3;        /* το g παραμένει 3.2 */
```



Οι δείκτες πρέπει να δηλώνονται

Το απόσπασμα προγράμματος αυτού θα μεταγλωττισθεί χωρίς σφάλματα αλλά δε θα δώσει το επιθυμητό αποτέλεσμα (γιατί;)

p1.c

```c
#include <stdio.h>

int main()
{
    float x = 12.35, y = -3.78;
    int *p;
    p = &x;
    y = *p;
    printf("The value of y is %.2f\n", y);
    return(0);
}
```

Οι δείκτες πρέπει να δηλώνονται

```
[Running] cd "/Users/eleftheriakatsiri/Downloads/lab_codes/" && gcc
p1.c -o p1 && "/Users/eleftheriakatsiri/Downloads/lab_codes/"p1
p1.c:7:7: warning: incompatible pointer types assigning to 'int *'
from 'float *' [-Wincompatible-pointer-types]
    p = &x;
      ^ ~~
1 warning generated.
The value of y is 1095080320.00

[Done] exited with code=0 in 1.229 seconds
```

Οι δείκτες πρέπει να δηλώνονται

Το απόσπασμα προγράμματος αυτού θα μεταγλωττισθεί χωρίς σφάλματα **και θα δώσει το επιθυμητό αποτέλεσμα** (γιατί;)

p2.c

```c
#include <stdio.h>

int main()
{
    float x = 12.35, y = -3.78;
    float *p;
    p = &x;
    y = *p;
    printf("The value of y is %.2f\n", y);
    return(0);
}
```

Οι δείκτες πρέπει να δηλώνονται

```
[Running] cd "/Users/eleftheriakatsiri/Downloads/lab_codes/" && gcc
p2.c -o p2 && "/Users/eleftheriakatsiri/Downloads/lab_codes/"p2
The value of y is 12.35

[Done] exited with code=0 in 1.373 seconds
```

Αριθμητικοί τελεστές

Υπάρχουν μόνο δύο αριθμητικοί τελεστές οι οποίοι μπορούν να χρησιμοποιηθούν

Ο + και ο -

Έστω ότι p1 είναι ο δείκτης ενός βραχέος ακεραίου με τρέχουσα τιμή 2000. Μετά την εντολή :

p1++;

το περιεχόμενο του p1 θα είναι 2002, όχι 2001 (γιατί;)

Η έκφραση: p1 = p1+ 9;

θα κάνει το p1 να δείξει στο ένατο στοιχείο του τύπου p1, πιο πέρα απ' αυτό στο οποίο δείχνει τώρα.

Τελεστές δεικτών

Δε μπορούμε δύο δείκτες :

1. **να τους πολλαπλασιάσουμε**
2. **να τους διαιρέσουμε**
3. **να αφαιρέσουμε τον ένα από τον άλλο**
4. **να τους προσθέσουμε**
5. **να εφαρμόσουμε τη μετατόπιση bitwise**

Επίσης, δεν μπορούμε να προσθέσουμε ή να αφαιρέσουμε τύπους float ή double στους δείκτες. Μόνο ακεραίους!

Δύο δείκτες οι οποίοι αναφέρονται σε ξεχωριστούς τύπους μεταβλητών δεν έχουν σχέση μεταξύ τους.

Αρχικές τιμές δεικτών

- Όταν δηλωθεί ο δείκτης, και πριν χρησιμοποιηθεί, θα περιέχει μία τιμή χωρίς νόημα.
- Αν χρησιμοποιήσουμε ένα δείκτη πριν του δώσουμε μία τιμή, πιθανόν θα καταστρέψουμε το πρόγραμμά μας!
- Συμβατικά δίνουμε σε ένα δείκτη μία αρχική μηδενική τιμή, προκειμένου να φανεί ότι δεν δείχνει πουθενά.
- Ένας δείκτης που έχει τιμή μηδέν (NULL) δεν δείχνει σε κανένα σημείο κι έτσι είναι ελεύθερος για χρήση

Η συνάρτηση malloc( )

Η συνάρτηση malloc( ) χρησιμοποιείται για να ζητήσουμε (να δεσμεύσουμε) μια περιοχή της μνήμης (Heap).

π.χ.

```
int *p1;
p1 = malloc (50) ;
void *malloc( );
```

Η συνάρτηση malloc( ) επιστρέφει ένα δείκτη σε void, ή την τιμή NULL εάν δεν υπάρχει αρκετή διαθέσιμη μνήμη ή ακόμη εάν υπάρχει κάποιο άλλο λάθος

Η συνάρτηση free

Η συνάρτηση free( ) χρησιμοποιείται για να απελευθερωθεί η περιοχή της μνήμης η οποία έχει δεσμευτεί με τη συνάρτηση malloc( ) (περιοχή Heap).

Π.χ.

```
int * iptr = (int*) malloc(sizeof(int));
free(iptr);
```

Προσοχή ! Δεν πρέπει να απελευθερώνουμε την ίδια περιοχή μνήμης ΔΥΟ φορές.

Παράδειγμα

```
// memory management problems
#include <stdlib.h>
int main( )
{
      void *pMem=malloc(100);
      free(pMem);
      free(pMem);
      return (0);
}
```

Παράδειγμα

```
1   // memory management problems
2   #include <stdlib.h>
3   #include <stdio.h>
4   int main( )
5   {
6    void *pMem=malloc(100);
7    free(pMem);
8    free(pMem);
9    printf("hello");
10   return (0);
11  }
12
```

Έξοδος

```
[Running] cd "/Users/eleftheriakatsiri/Downloads/lab_codes/" && gcc p4.c -o p4 &&
"/Users/eleftheriakatsiri/Downloads/lab_codes/"p4
/bin/sh: line 1:  8292 Abort trap: 6           "/Users/eleftheriakatsiri/Downloads/
lab_codes/"p4

[Done] exited with code=134 in 1.297 seconds
```

Πίνακες

Λέμε **πίνακα** (***array***) μια δομή δεδομένων, στην οποία ένα σύνολο αντικειμένων **του ίδιου τύπου** αποθηκεύονται σειριακά (το ένα μετά το άλλο). Δηλώνουμε την παρουσία ενός πίνακα, γράφοντας:

```
int m[100];    /* Δήλωση πίνακα 100 θέσεων */
```

Ο δείκτης του πρώτου στοιχείου ενός πίνακα είναι πάντα το **μηδέν, δηλαδή,** αν γράψουμε:

```
int m[2];
```

αυτό σημαίνει ότι δεσμεύουμε δύο θέσεις μνήμης, τις :

m[0] και m[1]

Ιδιότητες ενός πίνακα

- Έστω οι εντολές:

```
char str [80];
char *p1;
p1 = str;
```

- Το p1 λαμβάνει τη διεύθυνση του πρώτου στοιχείου του πίνακα str
- Αν θέλουμε να προσεγγίσουμε το πέμπτο στοιχείο του πίνακα str θα μπορούσαμε να γράψουμε :

```
str [4]  γιατί;
```

ή

```
*(p1+4)
```

Δείκτες και πίνακες

- **Υπάρχει μία στενή σχέση ανάμεσα στους δείκτες και τους πίνακες.**

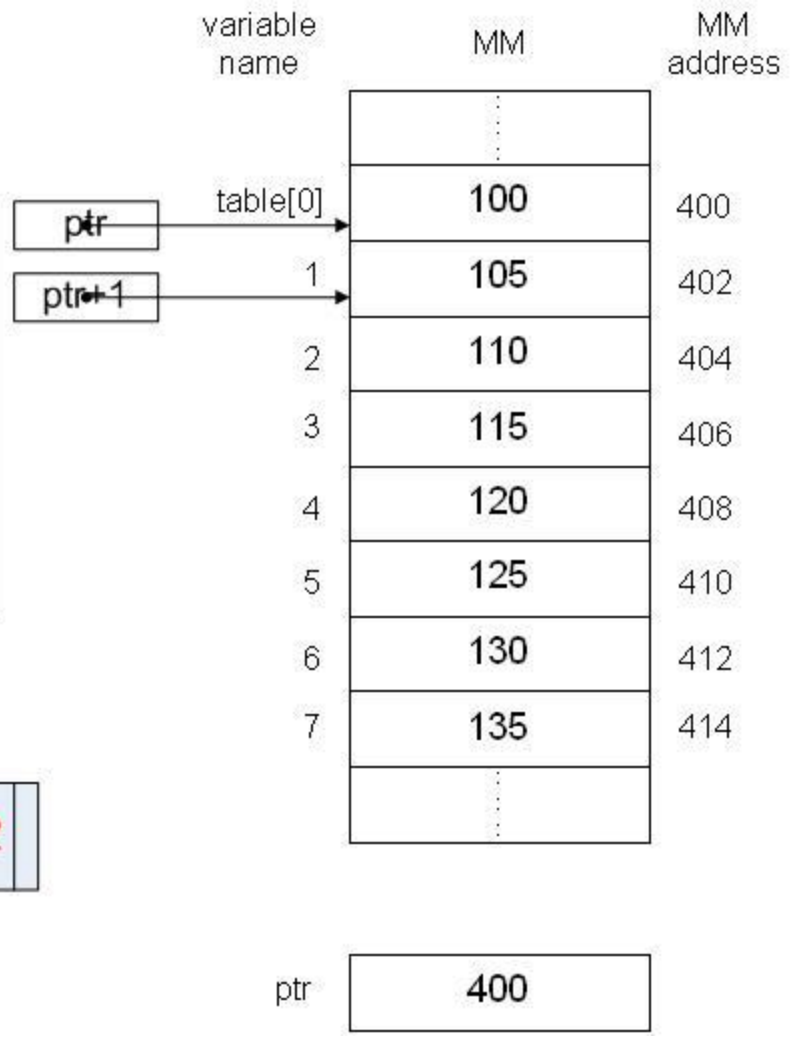
- **Η C επιτρέπει δύο μεθόδους προσέγγισης στοιχείων ενός πίνακα.**

- **Οι αριθμητικοί δείκτες μπορεί να είναι ταχύτεροι από την δεικτοποίηση του πίνακα.**

- **Επειδή η ταχύτητα είναι συχνά σημαντική στον προγραμματισμό, η χρήση των δεικτών στην εισαγωγή των στοιχείων πινάκων είναι πολύ διαδεδομένη στα προγράμματα της γλώσσας C.**

Παράδειγμα

```
short j;
short table[8];
short *ptr;
for ( j = 0; j <= 7; j++ )
   table[j] = 100 + j * 5;
ptr = &table[0];
```



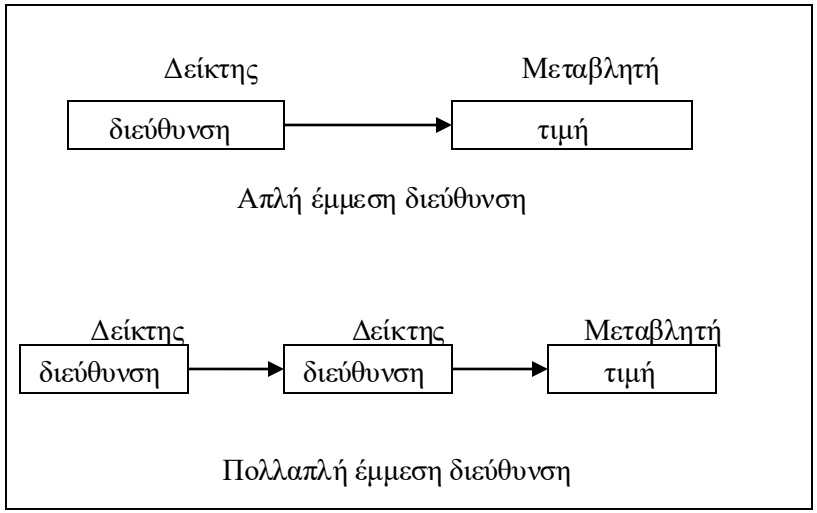
Note: the size of short is 2

Δύο εκδόσεις της συνάρτησης puts( )

A.
```
puts(s)
char s[];
{
 int t;
for(t=0; s[t]; ++t)
     putchar(s[t]);
}
```

B.
```
puts(s)
 char *s;
{
 while(*s)  putchar(*s++);
}
```

Δείκτες σε Δείκτες




```
int main( )
{
      int x, *p, **q;
      x = 10;
      p = &x;
      q = &p;
     printf("%d", **q);
      return(0);
 }
```

Προβλήματα με τους δείκτες

- **Οι δείκτες είναι ευχή αλλά και κατάρα.**
- **Προσφέρουν τεράστια δύναμη στον προγραμματιστή και είναι απαραίτητοι σε πολλά προγράμματα αλλά αν, παρ' ελπίδα, ένας δείκτης περιέχει μια λάθος τιμή μπορεί να γίνει το δυσκολότερο πρόβλημα που έχουμε να αντιμετωπίσουμε.**

```
int main( )
{
        int x, *p;
        x = 10;
        p = x;
        printf("%d",*p);
}
```

ΑΣΚΗΣΗ

Γράψτε μία συνάρτηση που θα ονομάζεται swapmin( ) και η οποία θα ανταλλάξει την τιμή δύο ακεραίων αν, και μόνον αν, η πρώτη παράμετρος είναι μικρότερη από τη δεύτερη.

```
swapmin(a,b)
 int  *a, *b;
  {   int  t;
        if (*a>= *b) return ;
        t = *a;
        *a = *b;
        *b = t;
   }
```

Οι πολυδιάστατοι πίνακες στη C

Ο γενικός τύπος δήλωσης ενός πολυδιάστατου πίνακα είναι :

τύπος όνομα[a][b][c]... [z];

Η αποθήκευση όλων των στοιχείων του πίνακα δεσμεύεται μονίμως στη μνήμη όσο χρόνο διαρκεί η εκτέλεση του προγράμματος.

Στη περίπτωση ενός δισδιάστατου πίνακα, ο αριθμός των bytes της μνήμης που απαιτούνται:

σειρές*στήλες*αριθμό των bytes του τύπου των δεδομένων

Έτσι για έναν πίνακα τιμών τύπου float με διαστάσεις 10,5 θα έχουν δεσμευθεί 10 * 5 * 4 δηλαδή 200 bytes μνήμης.

Οι πολυδιάστατοι πίνακες στη C

Όταν περνάμε δισδιάστατους πίνακες σε συναρτήσεις, περνάμε μόνον το δείκτη του πρώτου στοιχείου του πίνακα.

Αυτό μπορούμε να το κάνουμε χρησιμοποιώντας το όνομα του πίνακα χωρίς δείκτες.

Όμως, μία συνάρτηση η οποία δέχεται έναν δισδιάστατο πίνακα σαν παράμετρο πρέπει να γνωρίζει και το μέγεθος της δεύτερης διάστασης.

Π.χ. η συνάρτηση func1( ) η οποία δέχεται ένα δισδιάστατο πίνακα ακεραίων 30,10 πρέπει να δηλωθεί ως εξής:

```
func1(x)
int  x [ ] [10];
{          . . . . . .
}
```

Αρχικές τιμές

Για να δώσουμε αρχικές τιμές σ' ένα πίνακα 2 διαστάσεων μπορούμε να γράψουμε:

```
int a[4] [5]={
                {0, 1, 2, 0, 0},
                {2, 3, 4, 0, 0},
                {1, 0, 7, 0, 0},
                {6, 5, 8, 0, 0}
              };
```

ΑΣΚΗΣΗ

Γράψτε μία συνάρτηση η οποία θα συμπληρώνει με δυο τρόπους έναν πίνακα χαρακτήρων με τα γράμματα a,b,… έως k.

Ο πρώτος τρόπος να χρησιμοποιεί δεικτοποίηση πινάκων και ο δεύτερος δείκτες.

ΛΥΣΗ

με δεικτοποίηση

```
. load1( )
{ int t;
  for(t=0; t<10; ++t) a[t] = 'a' + t;
}
```

με δείκτες

```
   load2( )
{ int t;
  char *p;
  p=a;
  for( t=0; t<10; ++t) *p++ = 'a' + t;
}
```

Πίνακες Δεικτών

Οι δείκτες μπορούν να πινακοποιηθούν όπως γίνεται με οποιοδήποτε άλλο τύπο δεδομένων.
Οι μόνες τιμές τις οποίες μπορούν να κρατήσουν τα στοιχεία του πίνακα είναι οι διευθύνσεις των μεταβλητών.

Αν θέλουμε να περάσουμε έναν πίνακα δεικτών σε μία συνάρτηση, απλώς καλούμε τη συνάρτηση με το όνομα του πίνακα χωρίς καθόλου δείκτες.

Μία συνηθισμένη χρήση των πινάκων δεικτών είναι να κρατάνε τους δείκτες σε περίπτωση εσφαλμένων μηνυμάτων

Μπορούμε να δημιουργήσουμε μία συνάρτηση η οποία θα εξάγει ένα μήνυμα βασισμένο στον ακέραιο αριθμό αυτού του μηνύματος.

```
serror(i)
 int t;
  {
      static char *err[ ] =
       {
          "syntax error",
          "parentheses expected",
          "undefined variable",
          "duplicate label name"
       };
     printf (err[i] ) ;
  }
```

ΑΣΚΗΣΗ

Να γραφτεί μια συνάρτηση change(x, y) η οποία θα αλλάζει την τιμή του x σε x+y και την τιμή του y σε x*y .

Προσοχή, τα x και y, double

Λύση

```
float change( x, y )
   double *x, *y;
 { double p;
    p= *y;
    *y= (*x)*(*y);
    *x= *x+p;
   return 1;
 }
```