Δείκτες και Μονοδιάστατοι Πίνακες

Μια διακήρυξη της μορφής:

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

προκαλεί δέσμευση τεσσάρων συνεχόμενων θέσεων μνήμης για την αποθήκευση των στοιχείων του μονοδιάστατου πίνακα με όνομα a. Οι διευθύνσεις αυτών των στοιχείων θα είναι: &a[0], &a[1], &a[2] και &a[3].

Το όνομα του πίνακα είναι ένας σταθερός δείκτης που περιέχει τη διεύθυνση του πρώτου στοιχείου του (a == &a[0]). Αυτό το επαληθεύουμε εύκολα με το ακόλουθο πρόγραμμα:

```
int main()
{
  int i, a[4]={0}, b[4];

  for(i=0; i<4; i++)
  {
    printf("&a[%d] = %u : a[%d] = %d, ",i,&a[i],i,a[i]);
    printf("&b[%d] = %u : b[%d] = %d\n",i,&b[i],i,b[i]);
  }
  printf("\na = %u, b = %u\n",a,b);
  return 0;
}
```

Δείκτες και Μονοδιάστατοι Πίνακες

Έξοδος προγράμματος:

```
&a[0] = 2293584 : a[0] = 0, &b[0] = 2293568 : b[0] = 2009095316
&a[1] = 2293588 : a[1] = 0, &b[1] = 2293572 : b[1] = 2008948848
&a[2] = 2293592 : a[2] = 0, &b[2] = 2293576 : b[2] = -1
&a[3] = 2293596 : a[3] = 0, &b[3] = 2293580 : b[3] = 2009055971

a = 2293584, b = 2293568
```

"σκουπίδια"

Προσπέλαση στοιχείων πίνακα

Η προσπέλαση του `i` στοιχείου ενός πίνακα `a` μπορεί να γίνει με δύο τρόπους:

α) με χρήση της αναφοράς `a[i]`, ή β) μέσω αριθμητικής δεικτών: `*(a+i)`

Παραδείγματα:

```
a[0] = 12;   <==>  *a = 12;
a[1] = 31;   <==>  *(a+1) = 31;
a[5] = a[3]; <==>  *(a+5) = *(a+3);
a[i] = a[i-1] + a[i-2]; <==> *(a+i) = *(a+i-1) + *(a+i-2);
```

```c
#include <stdio.h>

int main()
{
  int pin[4]={10,20,30,40}, *ptr1, i;
  double A[4]={0.1,0.2,0.3,0.4}, *ptr2;

  ptr1 = pin;  /* Εκχώρηση διεύθυνσης pin στον ptr1 */
  ptr2 = A;    /* Εκχώρηση διεύθυνσης A στον ptr2 */

  for(i=0; i<4; i++)
  {
    printf("ptr1+%d: %8u  %d  --  ",i,ptr1+i,*(ptr1+i));
    printf("ptr2+%d: %8u  %.1f\n",i,ptr2+i,*(ptr2+i));
  }

  return 0;
}
```

Έξοδος προγράμματος:

```
ptr1+0:  1245012  10  --  ptr2+0:  1244976  0.1
ptr1+1:  1245016  20  --  ptr2+1:  1244984  0.2
ptr1+2:  1245020  30  --  ptr2+2:  1244992  0.3
ptr1+3:  1245024  40  --  ptr2+3:  1245000  0.4
```

Διαφορά μεταξύ πίνακα και δείκτη

Η δήλωση `int a[100];` α) δημιουργεί έναν δείκτη με όνομα a, β) δεσμεύει 100 θέσεις μνήμης τύπου int, γ) αρχικοποιεί τον δείκτη στη διεύθυνση της πρώτης θέσης μνήμης και δ) δεν επιτρέπει την μεταβολή του κατά την εκτέλεση του προγράμματος (σταθερός δείκτης).

Η δήλωση `int *a;` δημιουργεί μια μεταβλητή δείκτη με όνομα a αλλά δεν τον αρχικοποιεί ούτε δεσμεύει άλλες θέσεις μνήμης.

Προσοχή!!

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

Για παράδειγμα, έστω ότι δηλώσαμε έναν πίνακα 100 ακεραίων:

```c
int i, a[100];
```

και έστω ότι δεν ήμασταν προσεκτικοί όταν αρχικοποιήσαμε τον πίνακα αυτό μέσα από ένα βρόχο:

```c
for(i=1; i<=100; i++)  /* αντί του for(i=0; i<100; i++) */
  a[i] = i*i;
```

με αποτέλεσμα κατά την τελευταία επανάληψη, η τιμή του i να είναι 100 και η εκχώρηση του 100^2 να γίνει στο `a[100]` αντί του `a[99]`. Αυτό επιτρέπεται στις περισσότερες εκδόσεις της C και σχεδόν πάντα έχει καταστροφικά αποτελέσματα γιατί μπορεί να γράψει πάνω σε άλλα δεδομένα του προγράμματός μας!!

Περισσότερα για μεταβίβαση πινάκων σε συναρτήσεις

```
#include <stdio.h>
void printarray(int p[]);
int main()
{
  int array[100];
  . . .
  printarray(array);
  return 0;
}

void printarray(int p[])  /* ή void printarray(int *p) */
{
  int i;
  for(i=0; i<100; i++)
    printf("%d\n",p[i]);   /* ή printf("%d\n",*(p+i)); */
}
```

Το όρισμα της συνάρτησης για μεταβίβαση πίνακα είναι απλός pointer. Αν δηλωθεί ως `void printarray(int p[100])`*, το 100 αγνοείται καθώς δεν επιτρέπεται η αντιγραφή ολόκληρου πίνακα και ο p ερμηνεύεται ως ένας απλός δείκτης σε ακέραιο τύπο δεδομένων.*

Συναρτήσεις που επιστρέφουν δείκτη

```
#include <stdio.h>
int *max(int *a, int *b);
int main()
{
  int x,y,*ptr;
  ptr = &x;
  *ptr = 100;
  y = 200;
  printf("x = %d, y = %d\n",x,y);
  *max(&x, &y) = 50;
  printf("x = %d, y = %d\n",x,y);
  *max(&x, &y) = 500;
  printf("x = %d, y = %d\n",x,y);
  return 0;
}

int *max(int *a, int *b)
{
  if (*a > *b) return a; else return b;
}
```

Έξοδος προγράμματος:

```
x = 100, y = 200
x = 100, y = 50
x = 500, y = 50
```

Δείκτες και Δισδιάστατοι Πίνακες

Έστω η δήλωση:

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

Η C μας παρέχει τις εξής δυνατότητες χειρισμού δισδιάστατων πινάκων:

α) `a == &a[0][0]` (δηλαδή, όνομα πίνακα == διεύθυνση 1ου στοιχείου της 1ης γραμμής του πίνακα)

β) `a[0] == &a[0][0]` (δ/νση 1ου στοιχείου 1ης γραμμής)

`a[1] == &a[1][0]` (δ/νση 1ου στοιχείου 2ης γραμμής)

`a[2] == &a[2][0]` (δ/νση 1ου στοιχείου 3ης γραμμής)

`a[3] == &a[3][0]` (δ/νση 1ου στοιχείου 4ης γραμμής)

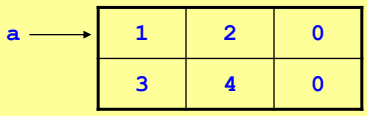
Γενικά,

`a[i] == &a[i][0],` δηλαδή, η διεύθυνση του 1ου στοιχείου της i-στής γραμμής του πίνακα.

Δείκτες και Δισδιάστατοι Πίνακες

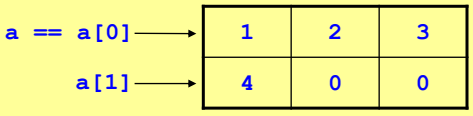
γ) 1ο παράδειγμα αρχικοποίησης 2-Δ πίνακα:

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



δ) 2ο παράδειγμα αρχικοποίησης 2-Δ πίνακα:

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



Δείκτες και Δισδιάστατοι Πίνακες

Ερμηνεία σύνθετων τύπων δεδομένων στη C που ορίζονται με τουλάχιστον 2 τελεστές. Οι κανόνες προτεραιότητας στις διακηρύξεις της C.

- **Κανόνας #1:** οι τελεστές παρενθέσεων ( ) και τετραγωνικών αγκυλών [ ] που αναφέρονται σε συναρτήσεις και πίνακες αντίστοιχα, βρίσκονται πάντα δεξιά του ονόματος (postfix operators). Η προτεραιότητά τους αυξάνεται αντιστρόφως ανάλογα της απόστασής τους από το όνομα της μεταβλητής.
- **Κανόνας #2:** ο τελεστής έμμεσης αναφοράς * που διαβάζεται "δείκτης σε" βρίσκεται πάντα αριστερά από το όνομα της μεταβλητής (prefix operator) και έχει χαμηλότερη προτεραιότητα έναντι των postfix τελεστών ανεξαρτήτως της απόστασής των από το όνομα της μεταβλητής.
- **Κανόνας #3:** αλλαγή προτεραιοτήτων, δηλαδή σειράς εφαρμογής των τελεστών, γίνεται με χρήση παρενθέσεων.

Παράδειγμα 1:

```
int a[10][100];
```

Ο πρώτος τελεστής (**[10]**), που είναι και ο πιο κοντινός, καθορίζει ότι η μεταβλητή **a** είναι πίνακας 10 στοιχείων ενώ ο δεύτερος τελεστής (**[100]**) μας λέει ότι κάθε ένα από τα στοιχεία του πίνακα **a** είναι ένας πίνακας 100 στοιχείων τύπου **int**.