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
```

Συναρτήσεις που επιστρέφουν δείκτη

```c
#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[3][4];
```

Η C μας παρέχει τις εξής δυνατότητες χειρισμού διδιάστατων 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)

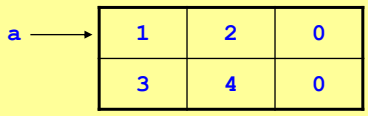
Γενικά,

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

Δείκτες και Δισδιάστατοι Πίνακες

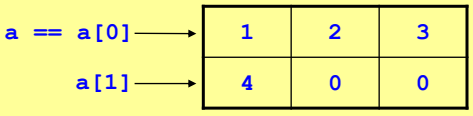
γ) 1ο παράδειγμα αρχικοποίησης 2-Δ πίνακα:

```
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];  /* op1: [10] ,  op2: [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**.

Συνεπώς, αφού ο **a** ερμηνεύεται ως μονοδιάστατος πίνακας τα στοιχεία του θα είναι τα **a[0], a[1], ..., a[9]** και επειδή αυτά είναι επίσης πίνακες, το κάθε ένα από αυτά, π.χ. το **a[i]**, στην ουσία θα είναι το όνομα ενός "εμφωλευμένου" πίνακα με στοιχεία τα **a[i][0], ..., a[i][99]** και άρα δείκτης στο πρώτο του στοιχείο (**a[i] == &a[i][0]**).

Αφού ο **a[i]** είναι πίνακας ακεραίων, ο **a[i]** θα δείχνει στο πρώτο στοιχείο του και ο (**a[i]+1**) θα δείχνει στο δεύτερο στοιχείο, δηλαδή στον επόμενο ακέραιο, αυξάνοντας τη διεύθυνση κατά 4 bytes. Αντίστοιχα, αφού ο **a** είναι πίνακας από πίνακες των 100 ακεραίων, ο **a** θα δείχνει στο πρώτο στοιχείο του που είναι ο πίνακας **a[0]** και ο (**a+1**) θα δείχνει στο επόμενο στοιχείο του, δηλαδή στον πίνακα **a[1]** προσπερνώντας τα 400 bytes συνεχόμενης μνήμης στα οποία αποθηκεύονται οι 100 ακέραιοι του πρώτου πίνακα.

Πρόγραμμα για εμφάνιση των διευθύνσεων στο παραπάνω παράδειγμα:

```
#include <stdio.h>
int main(void)
{
  int  j, a[10][100];
  printf("&j = %u, &j+1 = %u\n",&j,&j+1);
  printf("a = %u, a+1 = %u\n",a,a+1);
  printf("a[0] = %u, a[0]+1 = %u\n",a[0],a[0]+1);
  printf("&a[0][0] = %u, &a[0][0]+1 = %u\n",&a[0][0],&a[0][0]+1);
  return 0;
}
```

Έξοδος του προγράμματος:

```
&j = 2289324, &j+1 = 2289328   (&j : σταθερός δείκτης σε θέσεις ακεραίων)
a = 2289328, a+1 = 2289728     (a : σταθερός δείκτης σε πίνακες 100 ακεραίων)
a[0] = 2289328, a[0]+1 = 2289332  (a[0] : σταθερός δείκτης σε θέσεις ακεραίων)
&a[0][0] = 2289328, &a[0][0]+1 = 2289332  (&a[0][0] : ότι και με τον &j)
```

Στο παραπάνω παράδειγμα: sizeof a[0][0] = 4, sizeof a[0] = 400 και sizeof a = 4000 bytes.

Τέλος, αν θέλαμε να αρχικοποιήσουμε τον πίνακα μέσα από διπλό βρόχο:

```
...
int  i, j, a[10][100];
for(i=0; i<10; i++)
  for(j=0; j<100; j++)
    a[i][j] = 0;   /* ή  *(a[i]+j) = 0;  */
```

αλλά όχι: `*(a+i*100+j) = 0;`

Ο λόγος που το τελευταίο είναι λάθος είναι ότι ενώ το offset του (i,j) στοιχείου από το πρώτο στοιχείο είναι σωστό (i*100+j) ως προς το πλήθος των ακεραίων, ο δείκτης a δεν δείχνει σε ακέραιους αλλά σε πίνακες των 100 ακεραίων! Κατά συνέπεια, του λέμε να προχωρήσει τον δείκτη προσπερνώντας (i*100+j) * 100 θέσεις ακεραίων. Το σωστό, αν θέλουμε να χρησιμοποιήσουμε τον δείκτη **a** θα ήταν να αλλάξουμε (με type casting) τον τύπο των δεδομένων στα οποία δείχνει, σε απλούς ακέραιους (αντί για πίνακες):

```
*((int *)a+i*100+j) = 0;
```

Παράδειγμα 2:

```
int *a[10];  /* (unsigned)(a+1) – (unsigned)a = 4 bytes */
```

Ερμηνεία: ο a είναι πίνακας από 10 pointers σε ακέραιο.

Παράδειγμα 3:

```
int (*a)[10];  /* (unsigned)(a+1) – (unsigned)a = 40 bytes */
```

Ερμηνεία: ο a είναι pointer σε πίνακα 10 ακεραίων.

Παράδειγμα 4:

```
int *a[10][20];  /* (unsigned)(a+1) – (unsigned)a = 80 bytes */
```

Ερμηνεία: ο a είναι 2-Δ πίνακας 10 γραμμών και 20 στηλών από pointers σε int.

Παράδειγμα 5:

```
int (*a[10])[20];  /* (unsigned)(a+1) – (unsigned)a = 4 bytes */
```

Ερμηνεία: ο a είναι πίνακας 10 δεικτών σε πίνακα 20 ακεραίων.

Παράδειγμα 6:

```
int (*a)[10][20];  /* (unsigned)(a+1) – (unsigned)a = 800 bytes */
```

Ερμηνεία: ο a είναι pointer σε 2-Δ 10 x 20 πίνακα ακεραίων.

Παράδειγμα αρχικοποίησης πίνακα Α (MxN) από το πληκτρολόγιο:

```
/* Κλασικός τρόπος αρχικοποίησης από το πληκτρολόγιο*/

for(i=0; i<M; i++)
  for(j=0; j<N; j++)
  {
    printf("\nA[%d][%d] = ",i,j);
    scanf("%d",&A[i][j]);
  }


/* Αρχικοποίηση με χρήση δεικτών */

for(i=0; i<M; i++)
  for(j=0; j<N; j++)
  {
    printf("\nA[%d][%d] = ",i,j);
    scanf("%d",A[i]+j); /* ή scanf("%d",(int *)A+i*N+j) */
  }
```

Υπολογισμός ανάστροφου πίνακα B = A^T με χρήση δεικτών

```
#include <stdio.h>
#define M 2
#define N 3
int main()
{
  int  i, j;
  int  A[M][N]={{1,2,3}, {4,5,6}};
  int  B[N][M]; /* anastrofos */

  /* Υπολογισμός ανάστροφου πίνακα B */
  for(i=0; i<M; i++)
    for(j=0; j<N; j++)
      *(B[j]+i) = *(A[i]+j);

  /* Εμφάνιση ανάστροφου πίνακα */
  for(i=0; i<N; i++)
  { for(j=0; j<M; j++)
      printf("%d ",*(B[i]+j));
    printf("\n");
  }
  return 0;
}
```