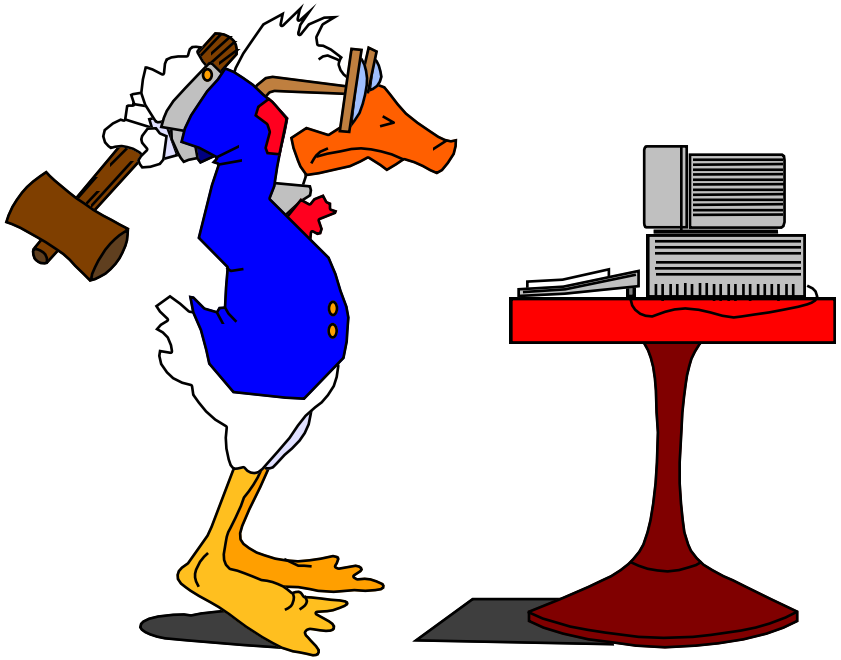
Οι δείκτες στη γλώσσα C

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

Σαν το ποδήλατο …

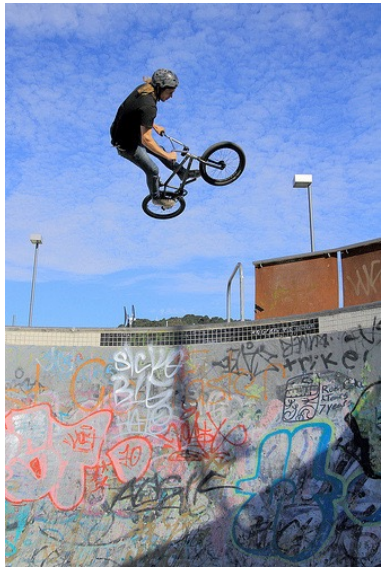
- **Κουράγιο:**

– 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.

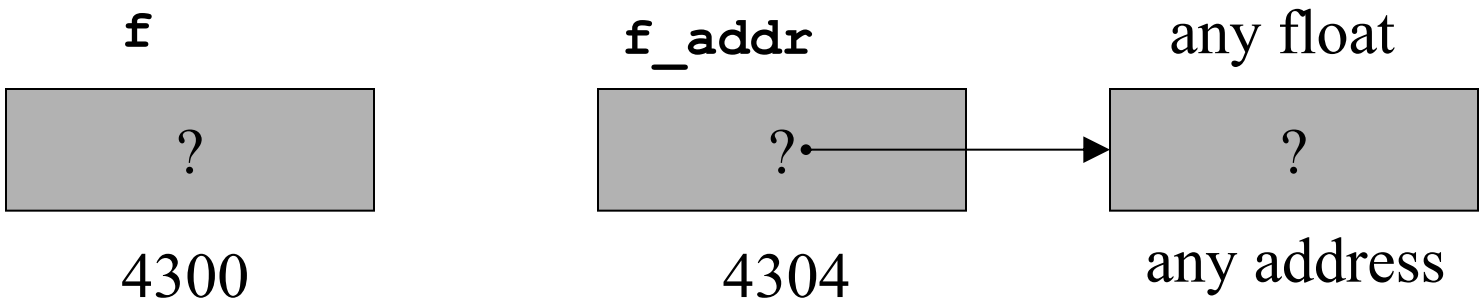
πριν

μετά

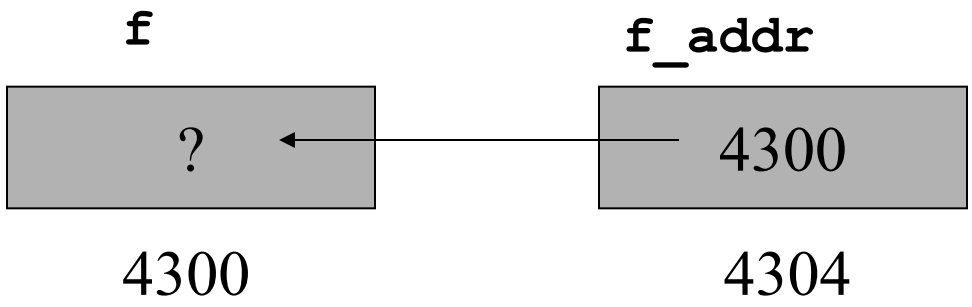


Δείκτες-Pointers (1)

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

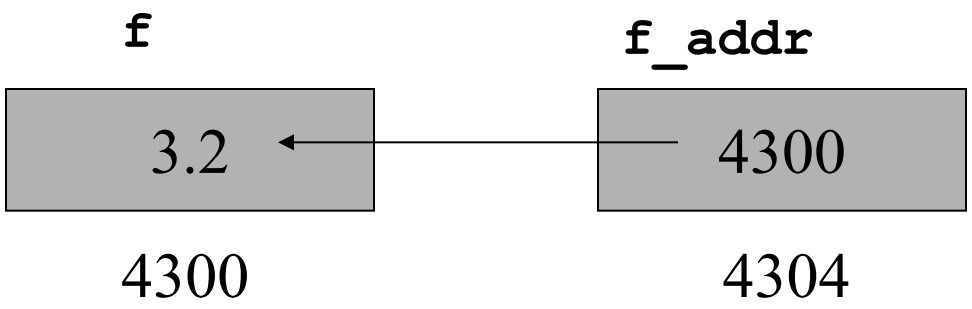



```
f_addr = &f;     /* & = τελεστής διεύθυνσης */
```

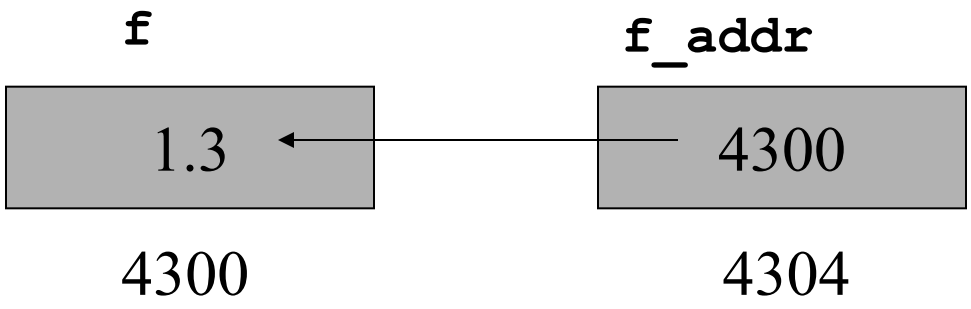


Δείκτες-Pointers (2)

```
*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

π.χ.

int *x;

float *y; (δείκτης μιας πραγματικής τιμής)

Το απόσπασμα προγράμματος που ακολουθεί θα μεταγλωττισθεί χωρίς σφάλματα αλλά δεν θα δώσει το επιθυμητό αποτέλεσμα (γιατί;)

```
float x, y;
x=12.35;
y=-3.78;
int *p;
p=&x;
y=*p;
```

Αριθμητικοί τελεστές

Υπάρχουν μόνο δύο αριθμητικοί τελεστές οι οποίοι μπορούν να χρησιμοποιηθούν

Ο + και ο -

Έστω ότι p1 είναι ο δείκτης ενός 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

- Δεν μπορούμε δύο δείκτες :

1. **να πολλαπλασιάσουμε**
2. **να διαιρέσουμε**
3. **ν' αφαιρέσουμε**
4. **να 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 όχι μόνο το πρόγραμμα μας αλλά ακόμη και το λειτουργικό σύστημα του υπολογιστή.

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;

}
```

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]

Αρχικοποίηση πίνακα με βρόχο επανάληψης

```
1  // Fig. 6.3: fig06_03.c
2  // Initializing the elements of an array to zeros.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void)
7  {
8     int n[5]; // n is an array of five integers
9
10    // set elements of array n to 0
11    for (size_t i = 0; i < 5; ++i) {
12       n[i] = 0; // set element at location i to 0
13    }
14
15    printf("%s%13s\n", "Element", "Value");
16
17    // output contents of array n in tabular format
18    for (size_t i = 0; i < 5; ++i) {
19       printf("%7u%13d\n", i, n[i]);
20    }
21 }
```

Αποτελέσματα

```
Element         Value
      0             0
      1             0
      2             0
      3             0
      4             0
```

Αρχικοποίηση πίνακα με χρήση λίστας αρχικοποίησης

```c
// Fig. 6.4: fig06_04.c
// Initializing the elements of an array with an initializer list.
#include <stdio.h>

// function main begins program execution
int main(void)
{
   // use initializer list to initialize array n
   int n[5] = {32, 27, 64, 18, 95};

   printf("%s%13s\n", "Element", "Value");

   // output contents of array in tabular format
   for (size_t i = 0; i < 5; ++i) {
      printf("%7u%13d\n", i, n[i]);
   }
}
```

Αποτελέσματα

```
Element         Value
      0            32
      1            27
      2            64
      3            18
      4            95
```

```
int n[10] = {0};
```

Arrays are not automatically initialised!

```
int n[] = {1, 2, 3, 4, 5};
```

```
int n[3] = {32, 27, 64, 18}; is a syntax error
```

Ορισμός πίνακα με χρήση συμβολικής σταθεράς και υπολογισμούς

```c
// Fig. 6.5: fig06_05.c
// Initializing the elements of array s to the even integers from 2 to 10.
#include <stdio.h>
#define SIZE 5 // maximum size of array

// function main begins program execution
int main(void)
{
   // symbolic constant SIZE can be used to specify array size
   int s[SIZE]; // array s has SIZE elements

   for (size_t j = 0; j < SIZE; ++j) { // set the values
      s[j] = 2 + 2 * j;
   }

   printf("%s%13s\n", "Element", "Value");

   // output contents of array s in tabular format
   for (size_t j = 0; j < SIZE; ++j) {
      printf("%7u%13d\n", j, s[j]);
   }
}
```

Αποτελέσματα

```
Element         Value
      0             2
      1             4
      2             6
      3             8
      4            10
```

- Μια συμβολική σταθερά είναι ένα αναγνωριστικό που αντικαθίσταται από το κείμενο αντικατάστασης από τον προεπεξεργαστή της C προτού το πρόγραμμα μεταγλωττιστεί.
- Όλες οι εμφανίσεις της σταθεράς SIZE αντικαθίστανται με το 5.
- Μια συμβολική σταθερά κάνει το πρόγραμμα πιο ευμετάβλητο.
- Π.χ αλλάζοντας το 5 με 1000 αρχικοποιούμε ένα πινακα 1000 γραμμών με μια αλλαγή (στην #define) ενώ γλυτώνουμε αλλαγές στις γραμμές 10,12,19 που θα κάναμε διαφορετικά!

Άθροισμα στοιχείων πίνακα

```
1  // Fig. 6.6: fig06_06.c
2  // Computing the sum of the elements of an array.
3  #include <stdio.h>
4  #define SIZE 12
5
6  // function main begins program execution
7  int main(void)
8  {
9     // use an initializer list to initialize the array
10    int a[SIZE] = {1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45};
11    int total = 0; // sum of array
12
13    // sum contents of array a
14    for (size_t i = 0; i < SIZE; ++i) {
15       total += a[i];
16    }
17
18    printf("Total of array element values is %d\n", total);
19 }
```

Αποτελέσματα

```
Total of array element values is 383
```

Σύνοψη απαντήσεων ερωτηματολογίου

- *Σαράντα φοιτητές ερωτήθηκαν σχετικά με την ποιότητα του φαγητού στο εστιατόριο.*
- *Τους ζητήθηκε να το βαθμολογήσουν από 1 μέχρι 10*
  - *(1 σημαίνει απαίσιο and 10 σημαίνει υπέροχο).*
- *Βάλτε τις 40 απαντήσεις σε ένα πίνακα ακεραίων και συνοψίστε τα αποτελέσματα του ερωτηματολογίου.*

Σύνοψη απαντήσεων ερωτηματολογίου

```c
// Fig. 6.7: fig06_07.c
// Analyzing a student poll.
#include <stdio.h>
#define RESPONSES_SIZE 40 // define array sizes
#define FREQUENCY_SIZE 11

// function main begins program execution
int main(void)
{
   // initialize frequency counters to 0
   int frequency[FREQUENCY_SIZE] = {0};

   // place the survey responses in the responses array
   int responses[RESPONSES_SIZE] = {1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10};

   // for each answer, select value of an element of array responses
   // and use that value as an index in array frequency to
   // determine element to increment
   for (size_t answer = 0; answer < RESPONSES_SIZE; ++answer) {
      ++frequency[responses[answer]];
   }

   // display results
   printf("%s%17s\n", "Rating", "Frequency");

   // output the frequencies in a tabular format
   for (size_t rating = 1; rating < FREQUENCY_SIZE; ++rating) {
      printf("%6d%17d\n", rating, frequency[rating]);
   }
}
```

Αποτελέσματα

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

Πίνακες χαρακτήρων

```
char string1[] = "first";
```

```
char string1[] = {'f', 'i', 'r', 's', 't', '\0'};
```

string1[0] = ‘f’

```
char string2[20];
scanf("%19s", string2);
```

Strings – char arrays

```c
// Fig. 6.10: fig06_10.c
// Treating character arrays as strings.
#include <stdio.h>
#define SIZE 20

// function main begins program execution
int main(void)
{
   char string1[SIZE]; // reserves 20 characters
   char string2[] = "string literal"; // reserves 15 characters

   // read string from user into array string1
   printf("%s", "Enter a string (no longer than 19 characters): ");
   scanf("%19s", string1); // input no more than 19 characters

   // output strings
   printf("string1 is: %s\nstring2 is: %s\n"
          "string1 with spaces between characters is:\n",
          string1, string2);

   // output characters until null character is reached
   for (size_t i = 0; i < SIZE && string1[i] != '\0'; ++i) {
      printf("%c ", string1[i]);
   }

   puts("");
}
```

Αποτέλεσμα

```
Enter a string (no longer than 19 characters): Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

Πέρασμα πίνακα σε συνάρτηση

```
int hourlyTemperatures[HOURS_IN_A_DAY];
```

```
modifyArray(hourlyTemperatures, HOURS_IN_A_DAY)
```

Πέρασμα πίνακα σε συνάρτηση

```
1  // Fig. 6.12: fig06_12.c
2  // Array name is the same as the address of the array's first element.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void)
7  {
8     char array[5]; // define an array of size 5
9
10    printf("    array = %p\n&array[0] = %p\n   &array = %p\n",
11       array, &array[0], &array);
12 }
```

Αποτελέσματα

```
    array = 0031F930
&array[0] = 0031F930
   &array = 0031F930
```

Πέρασμα πίνακα σε συνάρτηση (2)

```
1  // Fig. 6.13: fig06_13.c
2  // Passing arrays and individual array elements to functions.
3  #include <stdio.h>
4  #define SIZE 5
5
6  // function prototypes
7  void modifyArray(int b[], size_t size);
8  void modifyElement(int e);
9
10 // function main begins program execution
11 int main(void)
12 {
13    int a[SIZE] = {0, 1, 2, 3, 4}; // initialize array a
14
15    puts("Effects of passing entire array by reference:\n\nThe "
16       "values of the original array are:");
17
18    // output original array
19    for (size_t i = 0; i < SIZE; ++i) {
20       printf("%3d", a[i]);
21    }
22
23    puts(""); // outputs a newline
24
25    modifyArray(a, SIZE); // pass array a to modifyArray by reference
26    puts("The values of the modified array are:");
27
```

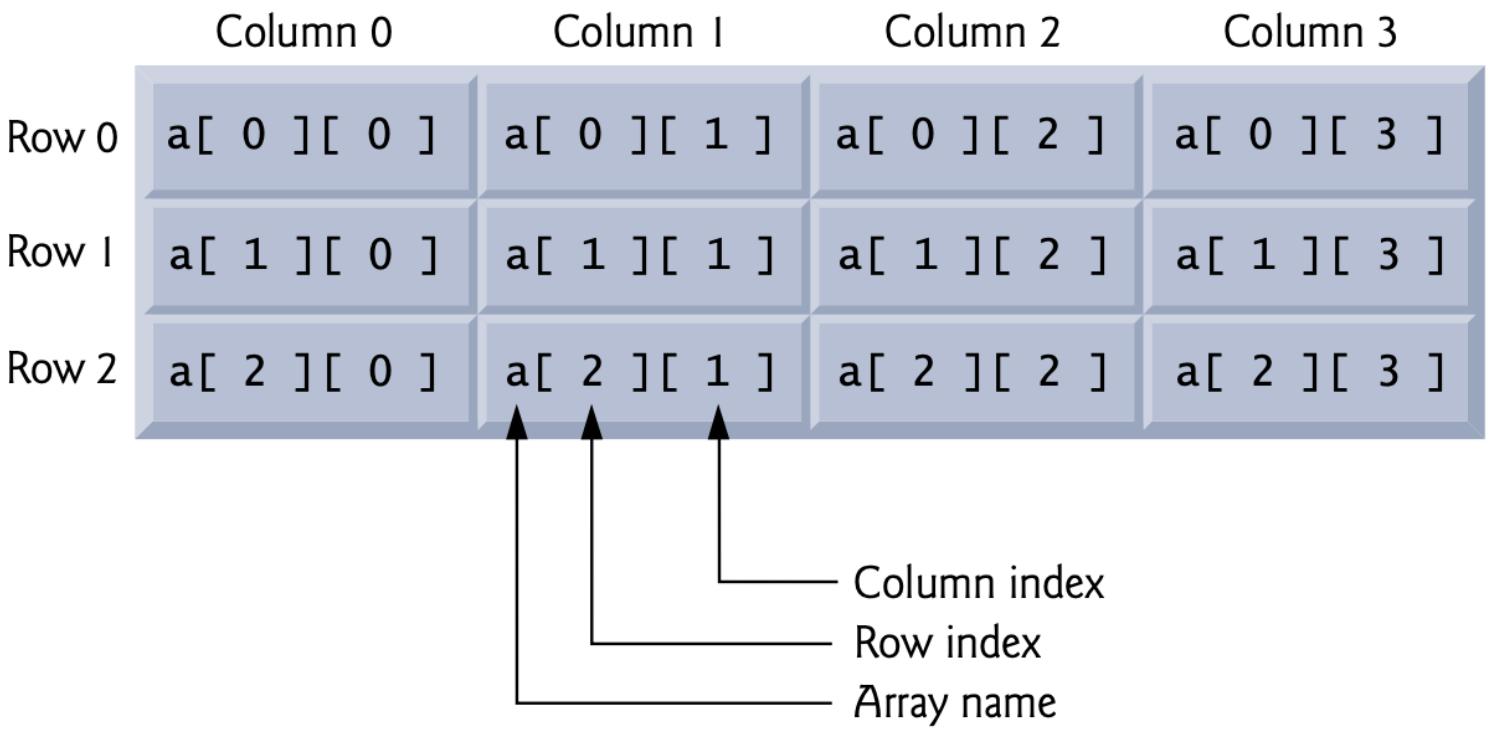
Πέρασμα πίνακα σε συνάρτηση (2)

```
28      // output modified array
29      for (size_t i = 0; i < SIZE; ++i) {
30         printf("%3d", a[i]);
31      }
32
33      // output value of a[3]
34      printf("\n\n\nEffects of passing array element "
35         "by value:\n\nThe value of a[3] is %d\n", a[3]);
36
37      modifyElement(a[3]); // pass array element a[3] by value
38
39      // output value of a[3]
40      printf("The value of a[3] is %d\n", a[3]);
41   }
42
43   // in function modifyArray, "b" points to the original array "a"
44   // in memory
45   void modifyArray(int b[], size_t size)
46   {
47      // multiply each array element by 2
48      for (size_t j = 0; j < size; ++j) {
49         b[j] *= 2; // actually modifies original array
50      }
51   }
52
53   // in function modifyElement, "e" is a local copy of array element
54   // a[3] passed from main
55   void modifyElement(int e)
56   {
57      // multiply parameter by 2
58      printf("Value in modifyElement is %d\n", e *= 2);
59   }
```

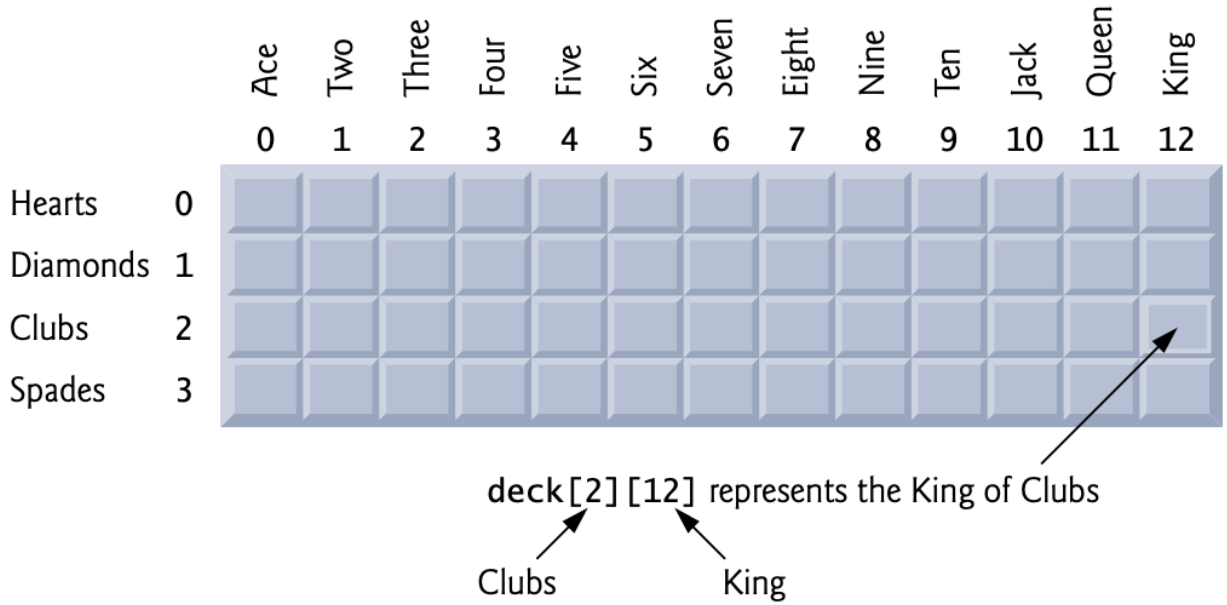
Αποτελέσματα

```
Effects of passing entire array by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element by value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

Πολυδιάστατοι πίνακες



Τράπουλα


Ace
Two
Three
Four
Five
Six
Seven
Eight
Nine
Ten
Jack
Queen
King
0 1 2 3 4 5 6 7 8 9 10 11 12
Hearts 0
Diamonds 1
Clubs 2
Spades 3
deck[2][12] represents the King of Clubs
Clubs
King

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.

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}
              };
```

Αρχικοποίηση

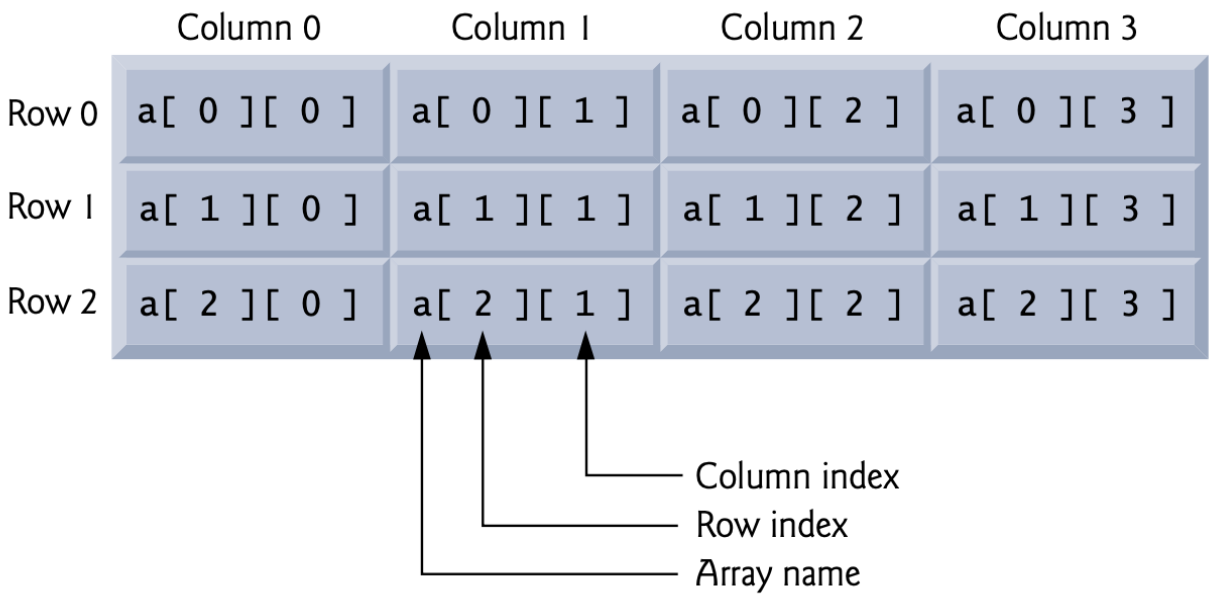
```
int b[2][2] = {{1, 2}, {3, 4}};
```

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

Αρχικοποίηση γραμμής

```
for (column = 0; column <= 3; ++column) {
   a[2][column] = 0;
}
```

```
a[2][0] = 0;
a[2][1] = 0;
a[2][2] = 0;
a[2][3] = 0;
```



Άθροισμα στοιχείων

```
total = 0;

for (row = 0; row <= 2; ++row) {
   for (column = 0; column <= 3; ++column) {
      total += a[row][column];
   }
}
```

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];
{              . . . . . .
 }
```

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

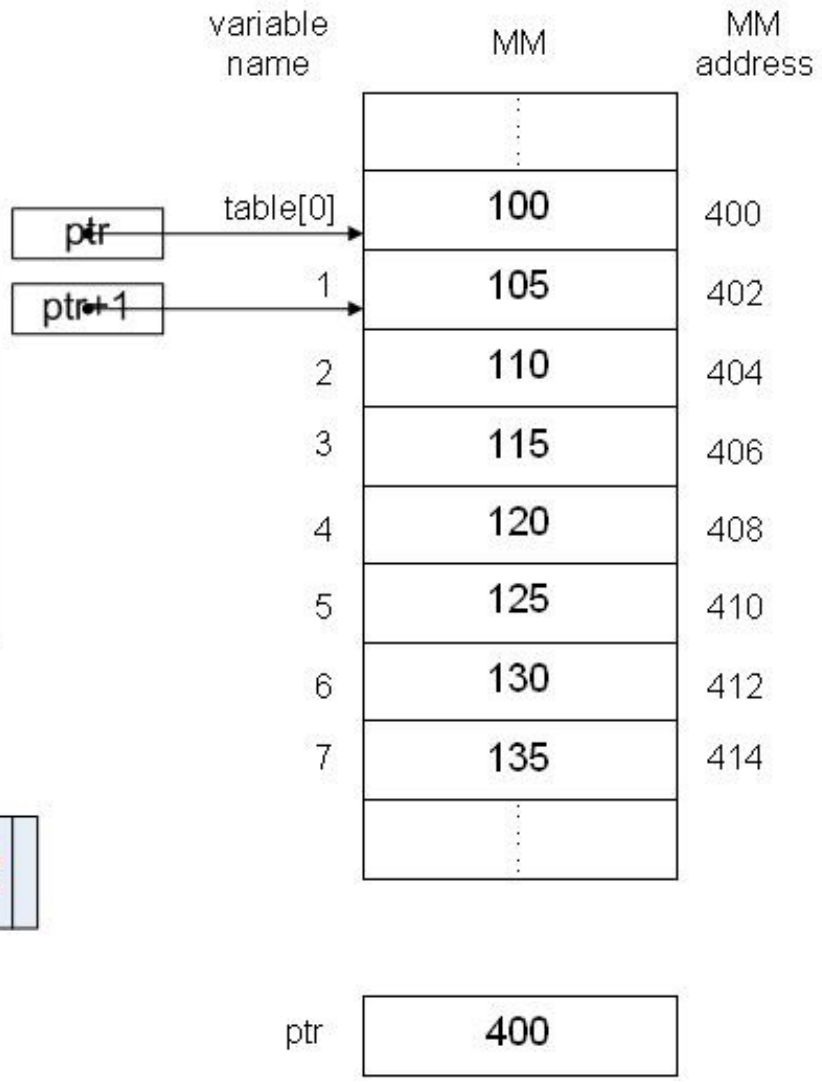
```
*(p1+4)
```

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

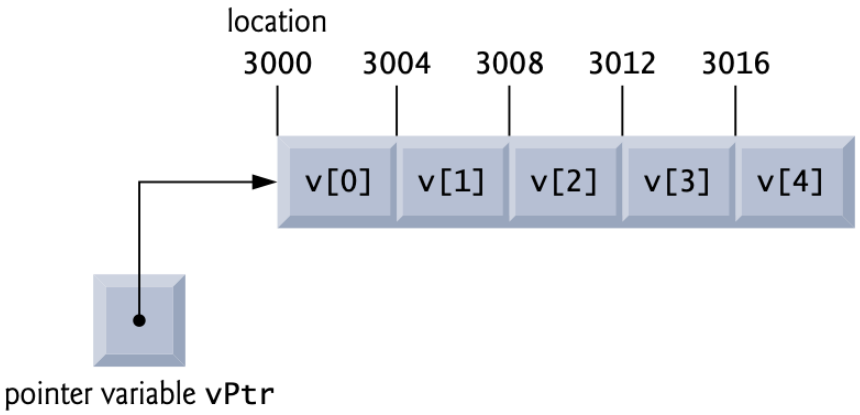
```
short j;
short table[8];
short *ptr;
for ( j = 0; j <= 7; j++ )
   table[j] = 100 + j * 5;
ptr = &table[0];
```



Note: the size of short is 2

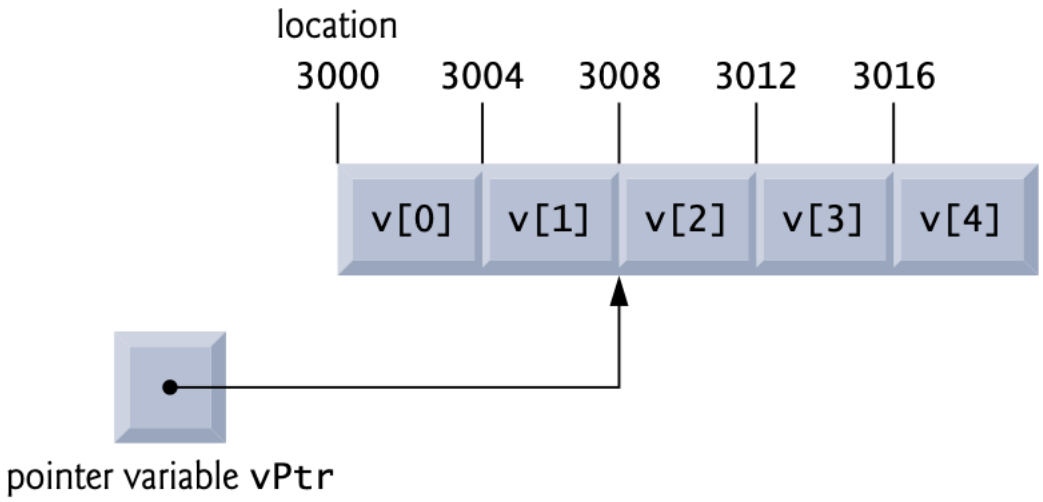
Στόχευση δείκτη σε πίνακα

```
vPtr = v;
vPtr = &v[0];
```



Πρόσθεση ακεραίου σε δείκτη

```
vPtr += 2;
```




```
vPtr -= 4;
```

```
x = v2Ptr - vPtr;
```

τελεστής sizeof()

```c
// Fig. 7.16: fig07_16.c
// Applying sizeof to an array name returns
// the number of bytes in the array.
#include <stdio.h>
#define SIZE 20

size_t getSize(float *ptr); // prototype

int main(void)
{
   float array[SIZE]; // create array

   printf("The number of bytes in the array is %u"
          "\nThe number of bytes returned by getSize is %u\n",
          sizeof(array), getSize(array));
}

// return size of ptr
size_t getSize(float *ptr)
{
   return sizeof(ptr);
}
```

Σχέση Πινάκων και Δεικτών

```
1  // Fig. 7.20: fig07_20.cpp
2  // Using indexing and pointer notations with arrays.
3  #include <stdio.h>
4  #define ARRAY_SIZE 4
5
6  int main(void)
7  {
8     int b[] = {10, 20, 30, 40}; // create and initialize array b
9     int *bPtr = b; // create bPtr and point it to array b
10
11    // output array b using array index notation
12    puts("Array b printed with:\nArray index notation");
13
14    // loop through array b
15    for (size_t i = 0; i < ARRAY_SIZE; ++i) {
16       printf("b[%u] = %d\n", i, b[i]);
17    }
18
19    // output array b using array name and pointer/offset notation
20    puts("\nPointer/offset notation where\n"
21          "the pointer is the array name");
22
23    // loop through array b
24    for (size_t offset = 0; offset < ARRAY_SIZE; ++offset) {
25       printf("*(b + %u) = %d\n", offset, *(b + offset));
26    }
```

Σχέση Πινάκων και Δεικτών 2

```
27
28      // output array b using bPtr and array index notation
29      puts("\nPointer index notation");
30
31      // loop through array b
32      for (size_t i = 0; i < ARRAY_SIZE; ++i) {
33         printf("bPtr[%u] = %d\n", i, bPtr[i]);
34      }
35
36      // output array b using bPtr and pointer/offset notation
37      puts("\nPointer/offset notation");
38
39      // loop through array b
40      for (size_t offset = 0; offset < ARRAY_SIZE; ++offset) {
41         printf("*(bPtr + %u) = %d\n", offset, *(bPtr + offset));
42      }
43   }
```

Αποτέλεσμα

```
Array b printed with:
Array index notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40

Pointer/offset notation where
the pointer is the array name
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

Pointer index notation
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

Αντιγραφή αλφαριθμητικών

```
1  // Fig. 7.21: fig07_21.c
2  // Copying a string using array notation and pointer notation.
3  #include <stdio.h>
4  #define SIZE 10
5
6  void copy1(char * const s1, const char * const s2); // prototype
7  void copy2(char *s1, const char *s2); // prototype
8
9  int main(void)
10 {
11    char string1[SIZE]; // create array string1
12    char *string2 = "Hello"; // create a pointer to a string
13
14    copy1(string1, string2);
15    printf("string1 = %s\n", string1);
16
17    char string3[SIZE]; // create array string3
18    char string4[] = "Good Bye"; // create an array containing a string
19
20    copy2(string3, string4);
21    printf("string3 = %s\n", string3);
22 }
23
```

Τρόπος α’

```
23
24  // copy s2 to s1 using array notation
25  void copy1(char * const s1, const char * const s2)
26  {
27     // loop through strings
28     for (size_t i = 0; (s1[i] = s2[i]) != '\0'; ++i) {
29        ; // do nothing in body
30     }
31  }
32
```

Τρόπος β’

```
32
33  // copy s2 to s1 using pointer notation
34  void copy2(char *s1, const char *s2)
35  {
36     // loop through strings
37     for (; (*s1 = *s2) != '\0'; ++s1, ++s2) {
38        ; // do nothing in body
39     }
40  }
```

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 πινακα /offset

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

με δείκτες / offset

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

Δύο εκδόσεις της συνάρτησης puts( )

A.

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

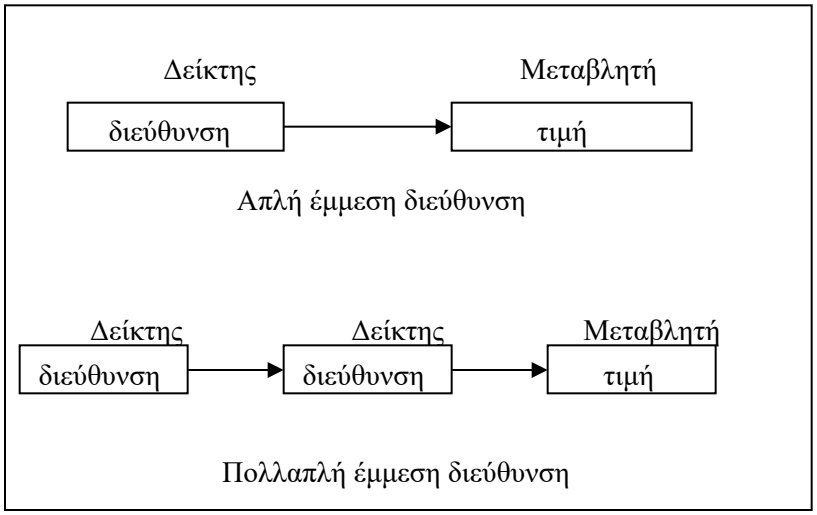
με δεικτοποίηση πινακα

B.

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

με δείκτες

Δείκτες σε Δείκτες




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

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.

Π.χ.

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

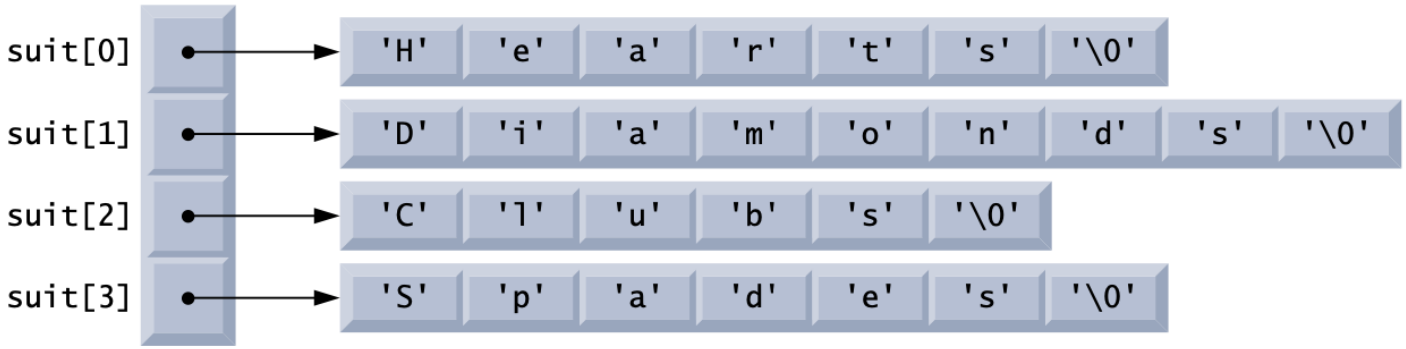
ΑΣΚΗΣΗ

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;
   }
```

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.

Πίνακες Δεικτών

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

```c
// Fig. 7.10: fig07_10.c
// Converting a string to uppercase using a
// non-constant pointer to non-constant data.
#include <stdio.h>
#include <ctype.h>

void convertToUppercase(char *sPtr); // prototype

int main(void)
{
   char string[] = "cHaRaCters and $32.98"; // initialize char array

   printf("The string before conversion is: %s", string);
   convertToUppercase(string);
   printf("\nThe string after conversion is: %s\n", string);
}

// convert string to uppercase letters
void convertToUppercase(char *sPtr)
{
   while (*sPtr != '\0') { // current character is not '\0'
      *sPtr = toupper(*sPtr); // convert to uppercase
      ++sPtr; // make sPtr point to the next character
   }
}
```

```
The string before conversion is: cHaRaCters and $32.98
The string after conversion is: CHARACTERS AND $32.98
```

```c
// Fig. 7.12: fig07_12.c
// Attempting to modify data through a
// non-constant pointer to constant data.
#include <stdio.h>
void f(const int *xPtr); // prototype

int main(void)
{
   int y; // define y

   f(&y); // f attempts illegal modification
}

// xPtr cannot be used to modify the
// value of the variable to which it points
void f(const int *xPtr)
{
   *xPtr = 100; // error: cannot modify a const object
}
```

```
error C2166: l-value specifies const object
```

```c
// Fig. 7.13: fig07_13.c
// Attempting to modify a constant pointer to non-constant data.
#include <stdio.h>

int main(void)
{
   int x; // define x
   int y; // define y

   // ptr is a constant pointer to an integer that can be modified
   // through ptr, but ptr always points to the same memory location
   int * const ptr = &x;

   *ptr = 7; // allowed: *ptr is not const
   ptr = &y; // error: ptr is const; cannot assign new address
}
```

```
c:\examples\ch07\fig07_13.c(15) : error C2166: l-value specifies const object
```