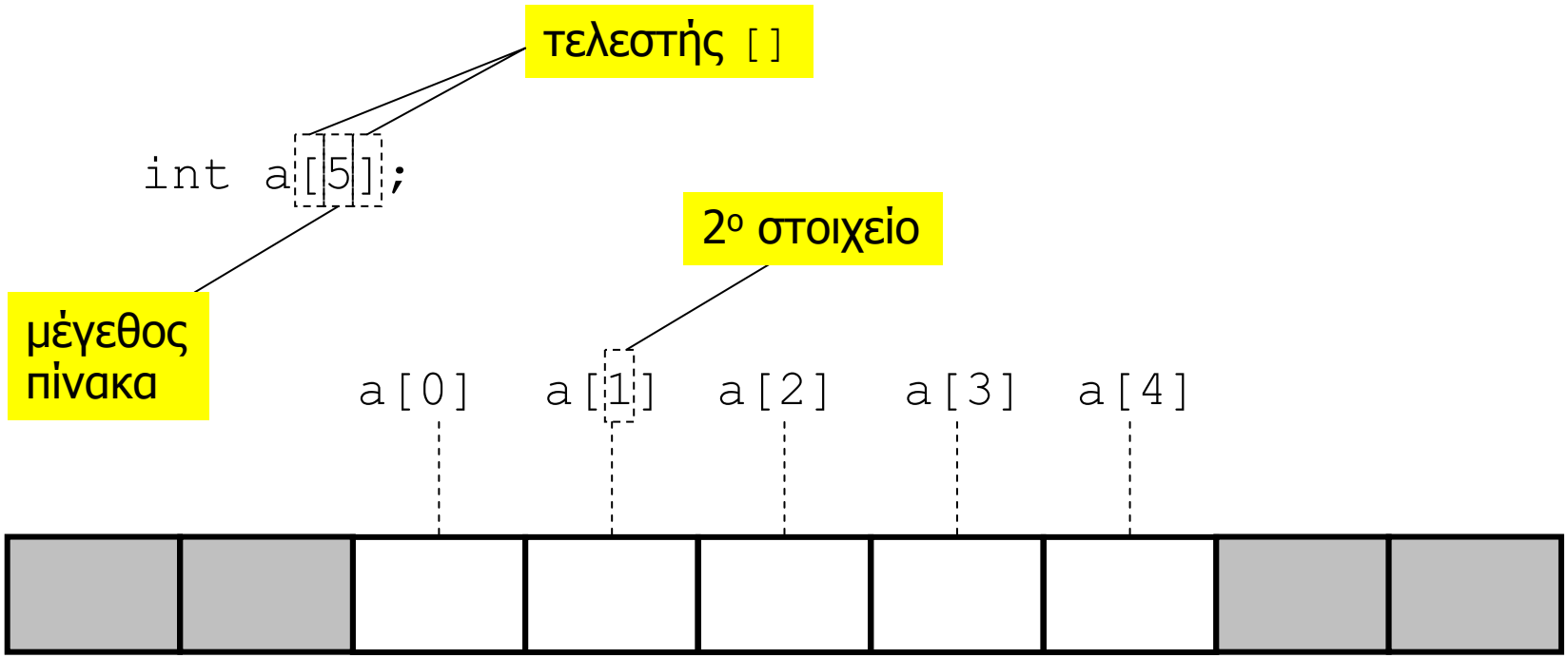
Πίνακες

Πίνακες

- Ο πίνακας είναι μια ειδική δομή για την αποθήκευση μιας **σειράς** από δεδομένα του **ίδιου** τύπου.
- Η δήλωση ενός πίνακα γίνεται όπως για μια κανονική μεταβλητή, σε συνδυασμό με τον τελεστή `[]` μέσω του οποίου δηλώνεται το μέγεθος του πίνακα.
- Το μέγεθος `N` του πίνακα δηλώνει τον αριθμό των **στοιχείων** του – στην ουσία, το ότι πρέπει να δεσμευτεί μνήμη για να αποθηκετούν `N` ξεχωριστές τιμές που αντιστοιχούν στον τύπο του.
- Κάθε στοιχείο ενός πίνακα από δεδομένα τύπου T είναι συντακτικά συμβατό με μια μεταβλητή τύπου T.
- Όπως μια συμβατική μεταβλητή, κάθε στοιχείο του πίνακα πρέπει να αρχικοποιηθεί με συγκεκριμένη τιμή.


τελεστής []
int a[5];
μέγεθος πίνακα
2ο στοιχείο
a[0]
a[1]
a[2]
a[3]
a[4]

Πρόσβαση σε στοιχεία του πίνακα

- Αν ο πίνακας έχει μέγεθος N (αντικείμενα), τότε η θέση `0` αντιστοιχεί στο πρώτο στοιχείο και η θέση `N-1` στο τελευταίο στοιχείο του πίνακα.
- Η μνήμη για την αποθήκευση των τιμών που θα δοθούν στα στοιχεία του πίνακα δεσμεύεται (από τον μεταφραστή) **συνεχόμενα**, δηλαδή η τιμή του στοιχείου στη θέση `i` αποθηκεύεται στην αμέσως επόμενη διεύθυνση από αυτή του στοιχείου `i-1`.
- Η πρόσβαση του πίνακα με τιμή θέσης **εκτός ορίων** αποτελεί προγραμματιστικό **λάθος** καθώς αντιστοιχεί σε πρόσβαση μνήμης που δεν ανήκει στον πίνακα.
- **Προσοχή:** ο μεταφραστής **δεν ελέγχει** αν η θέση που δίνει ο προγραμματιστής είναι εντός των ορίων.

```
int a;                  /* ακέραιος */
int b[3];               /* πίνακας 3 ακεραίων */
int c[] = {5,8,2};      /* πίνακας 3 ακεραίων,
                           με τιμές 5, 8, 2 */

a = b[0];               /* a γίνεται ? */

a = c[2];               /* a γίνεται 2 */

b[0] = c[1];            /* b[0] γίνεται 8 */

c[0]++;                 /* c[0] γίνεται 6 */

b[--a] = 3;             /* a γίνεται 1, b[1] γίνεται 3 */

a = c[-1];              /* !? */

b[3] = 5;               /* !? */
```

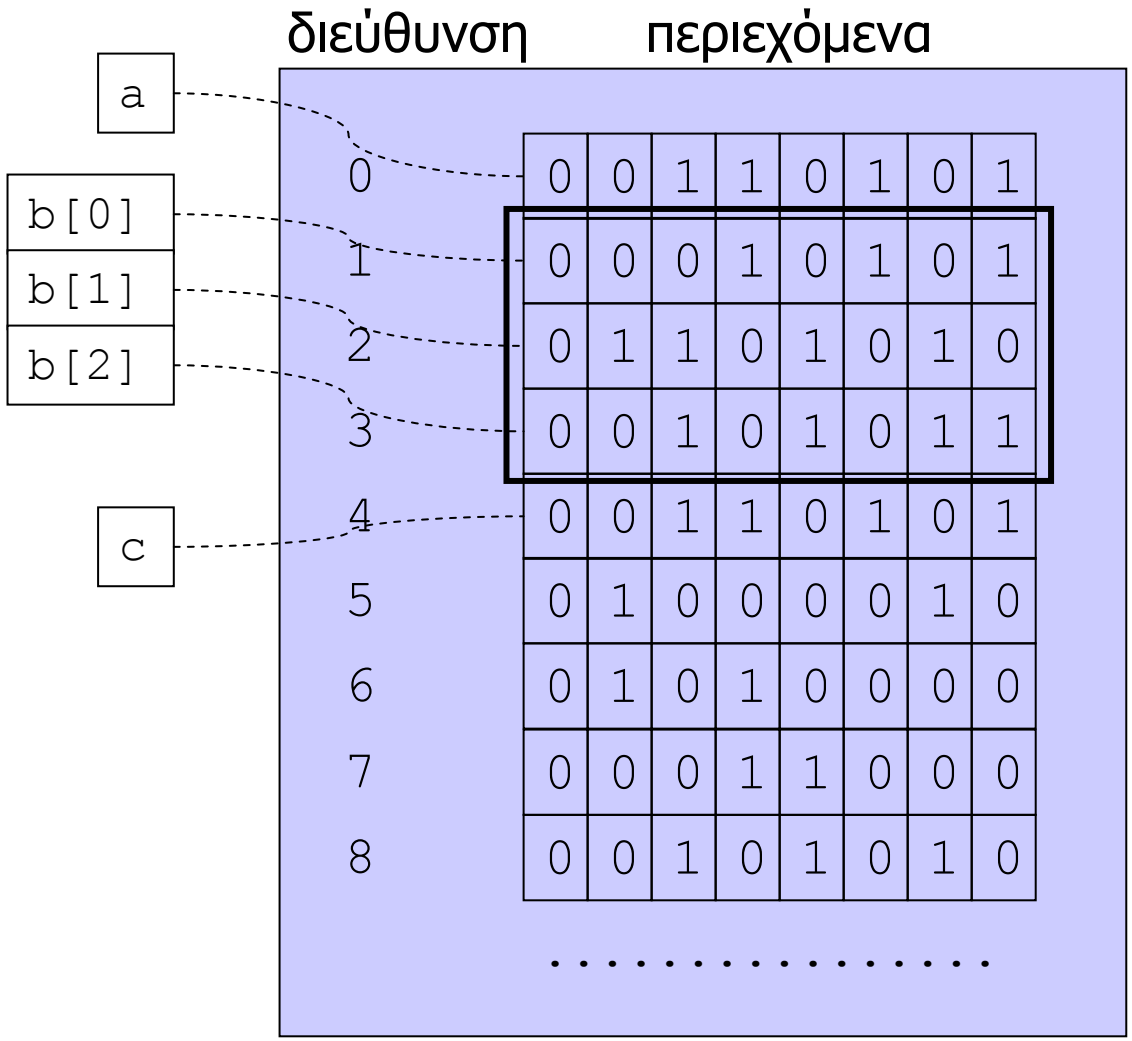
Δέσμευση μνήμη πίνακα

- Ο αριθμός των στοιχείων ενός πίνακα δίνεται (άμεσα ή έμμεσα) **κατά την δήλωση** του.
- Το μέγεθος του πίνακα πρέπει να είναι **γνωστό** και να δοθεί την ώρα της **συγγραφής** του κώδικα.
- Δεν υποστηρίζονται «ανοιχτοί» πίνακες, το μέγεθος των οποίων μπορεί να (επανα)προσδιοριστεί ή/και να αλλάξει κατά την διάρκεια της εκτέλεσης.
- Η μνήμη για την αποθήκευση των στοιχείων ενός πίνακα μεγέθους `N` από δεδομένα τύπου `T` είναι `N*sizeof(T)` και δεσμεύεται **μονομιάς** (ανεξάρτητα με το ποιά στοιχεία του πίνακα θα χρησιμοποιηθούν τελικά από το πρόγαμμα).

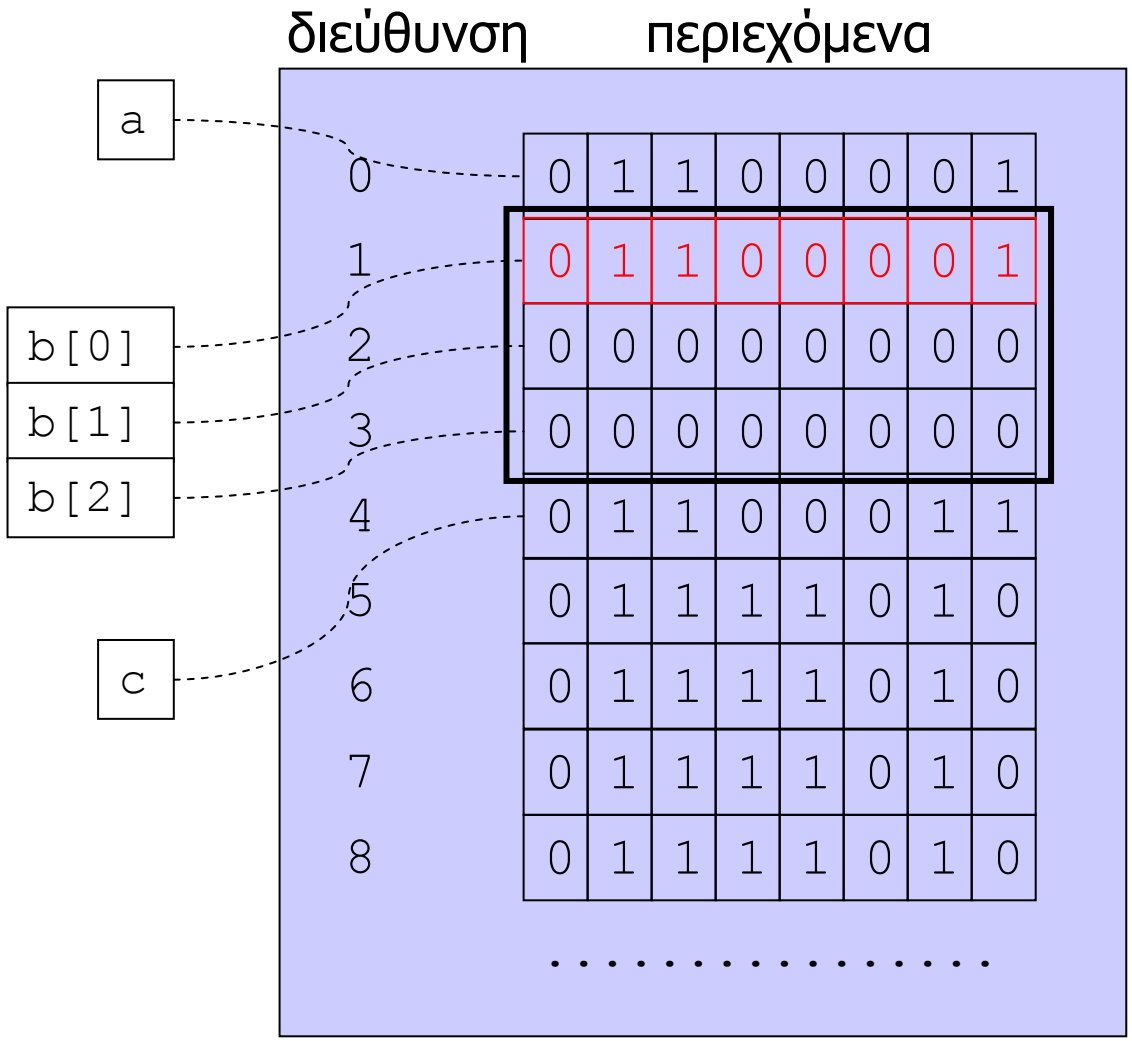
Ασφάλεια προσπέλασης μνήμης

- Με τις συμβατικές μεταβλητές ένα πρόγραμμα δεν μπορεί να προσπελάσει «λάθος» θέσεις μνήμης.
- **Αυτό δεν ισχύει όταν χρησιμοποιούμε πίνακες.**
- Η πρόσβαση σε θέση που βρίσκεται εκτός των ορίων του πίνακα **δεν** εντοπίζεται από τον μεταφραστή.
- **Ούτε** (απαραίτητα) κατά την εκτέλεση.
- Μπορεί (κατά λάθος) να διαβαστούν / γραφτούν τιμές σε θέσεις μνήμης **εκτός** της περιοχής του πίνακα με απροσδόκητα (λάθος) αποτελέσματα.
- **Σημείωση:** υπάρχει περίπτωση το πρόγραμμα να «καταστρέφει» δεδομένα **δικών του** μεταβλητών.

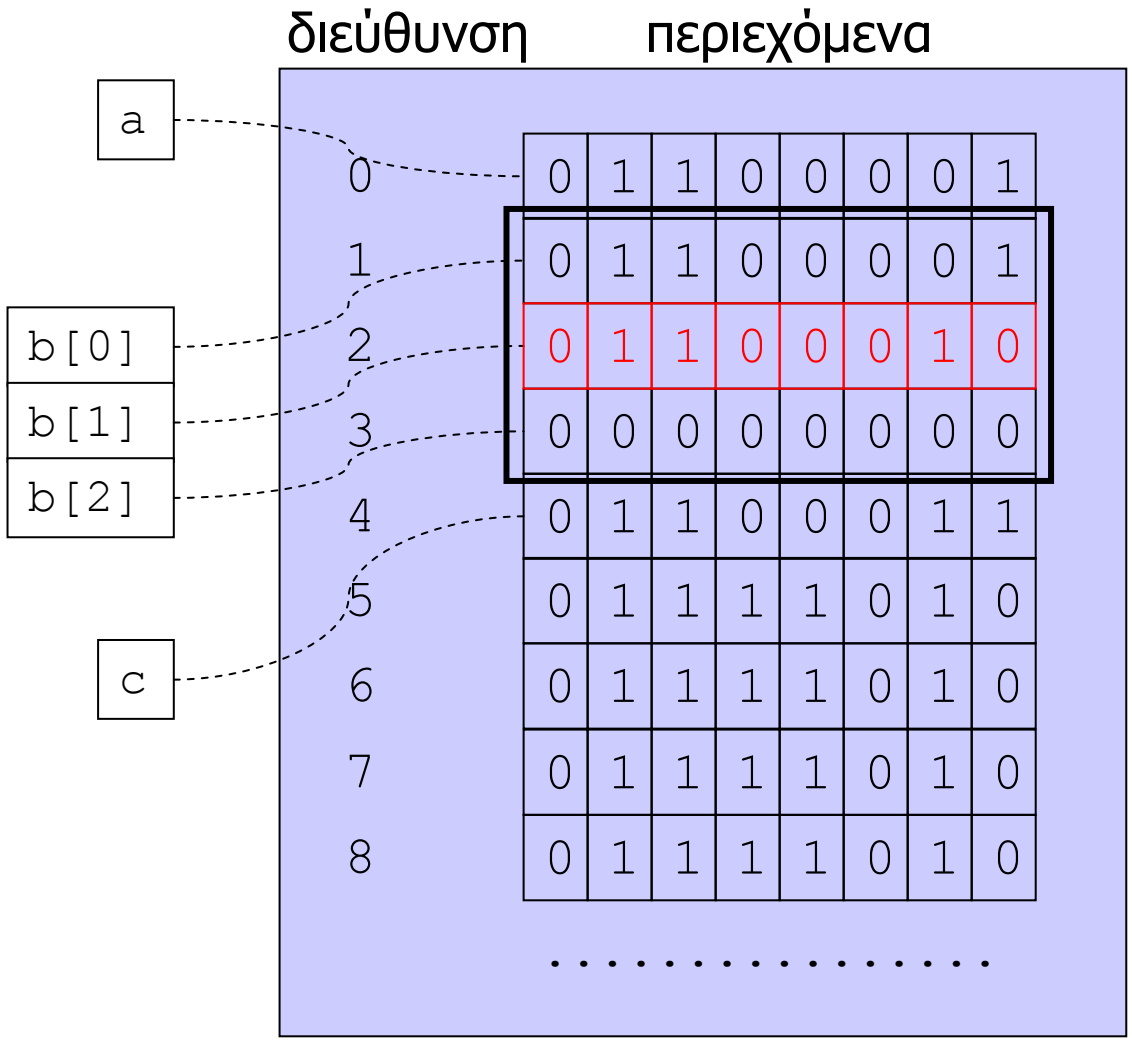
```
char a;
char b[3];
char c;
```



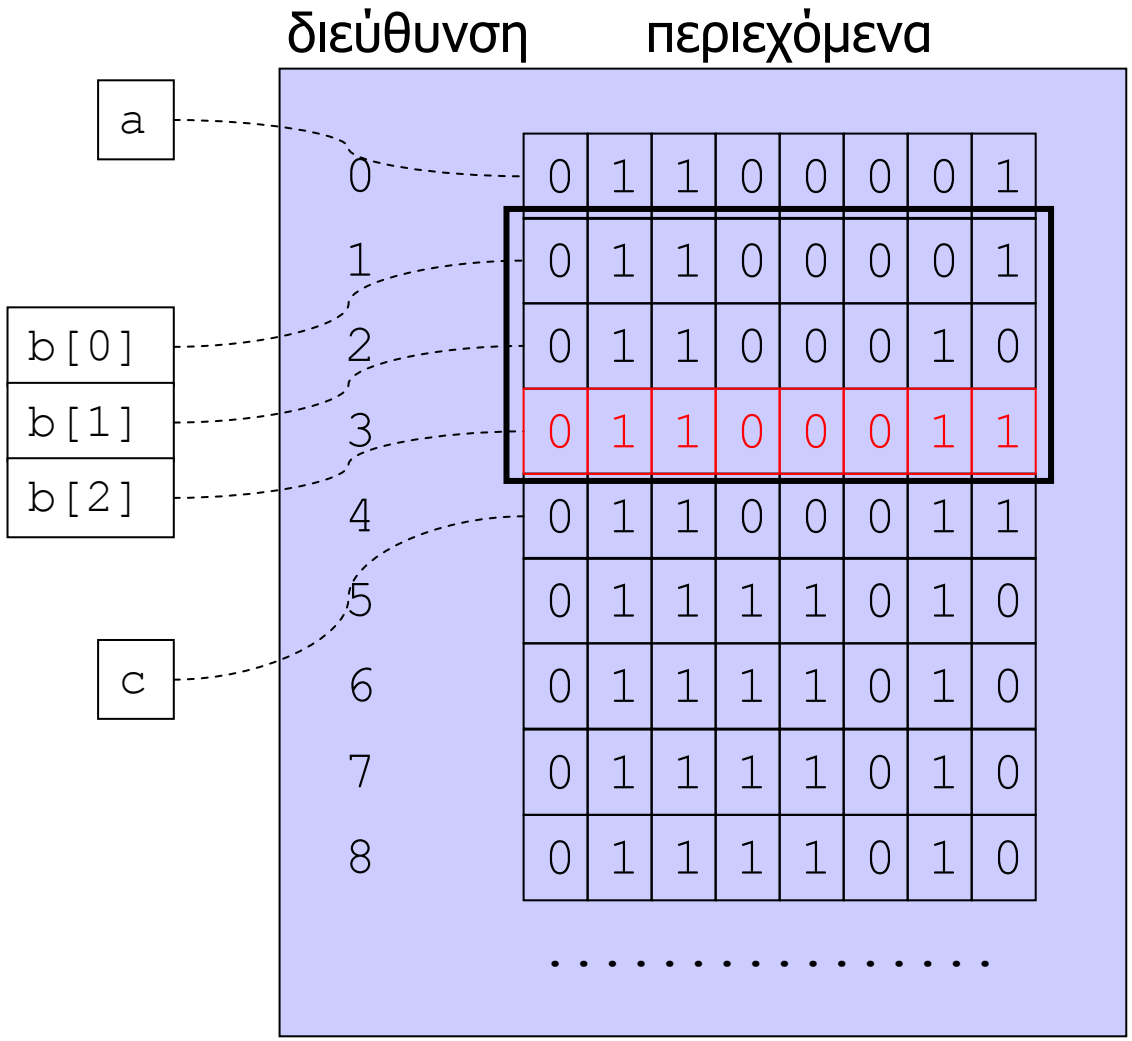
```
char a;
char b[3];
char c;

b[0]='a';
```

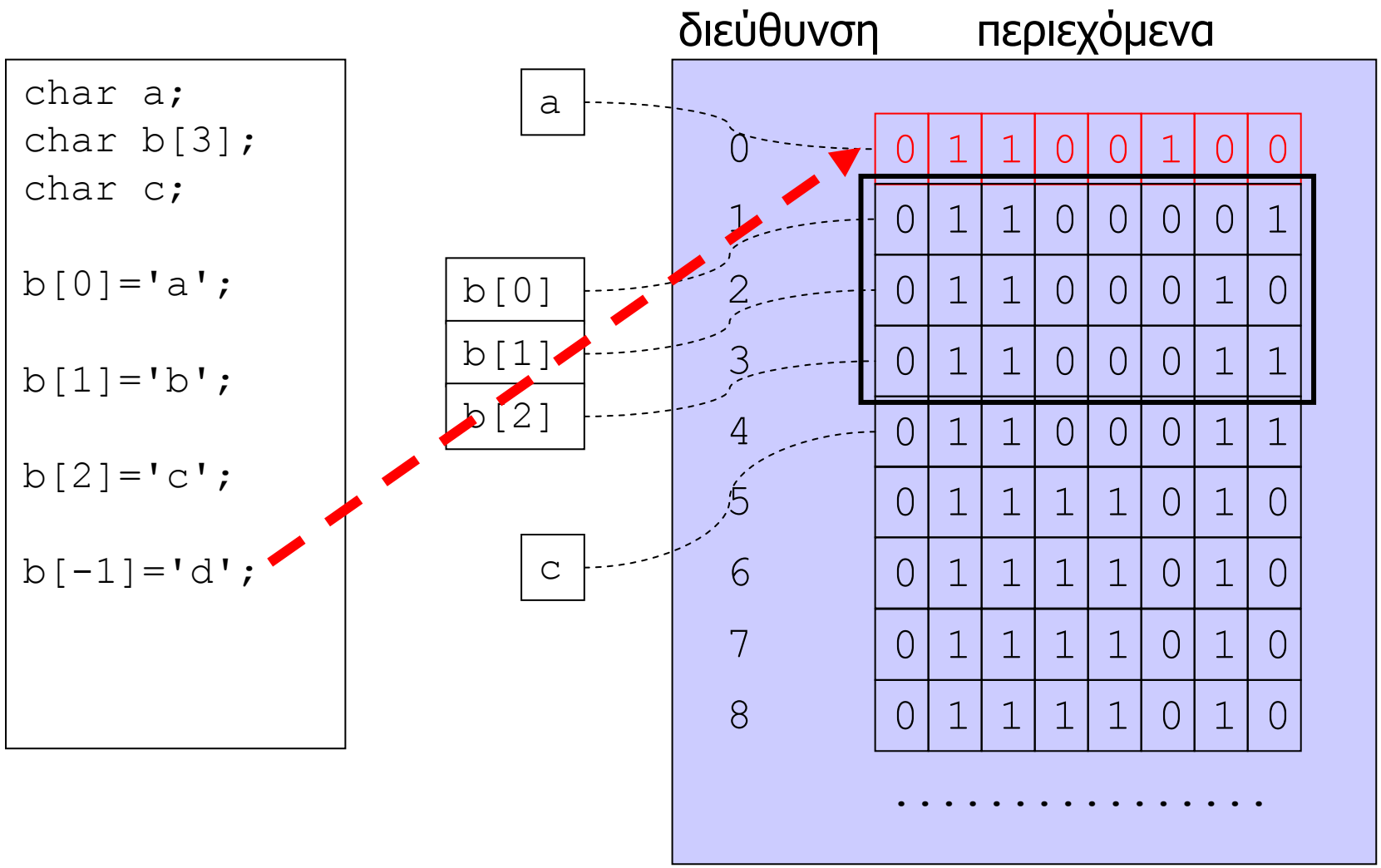


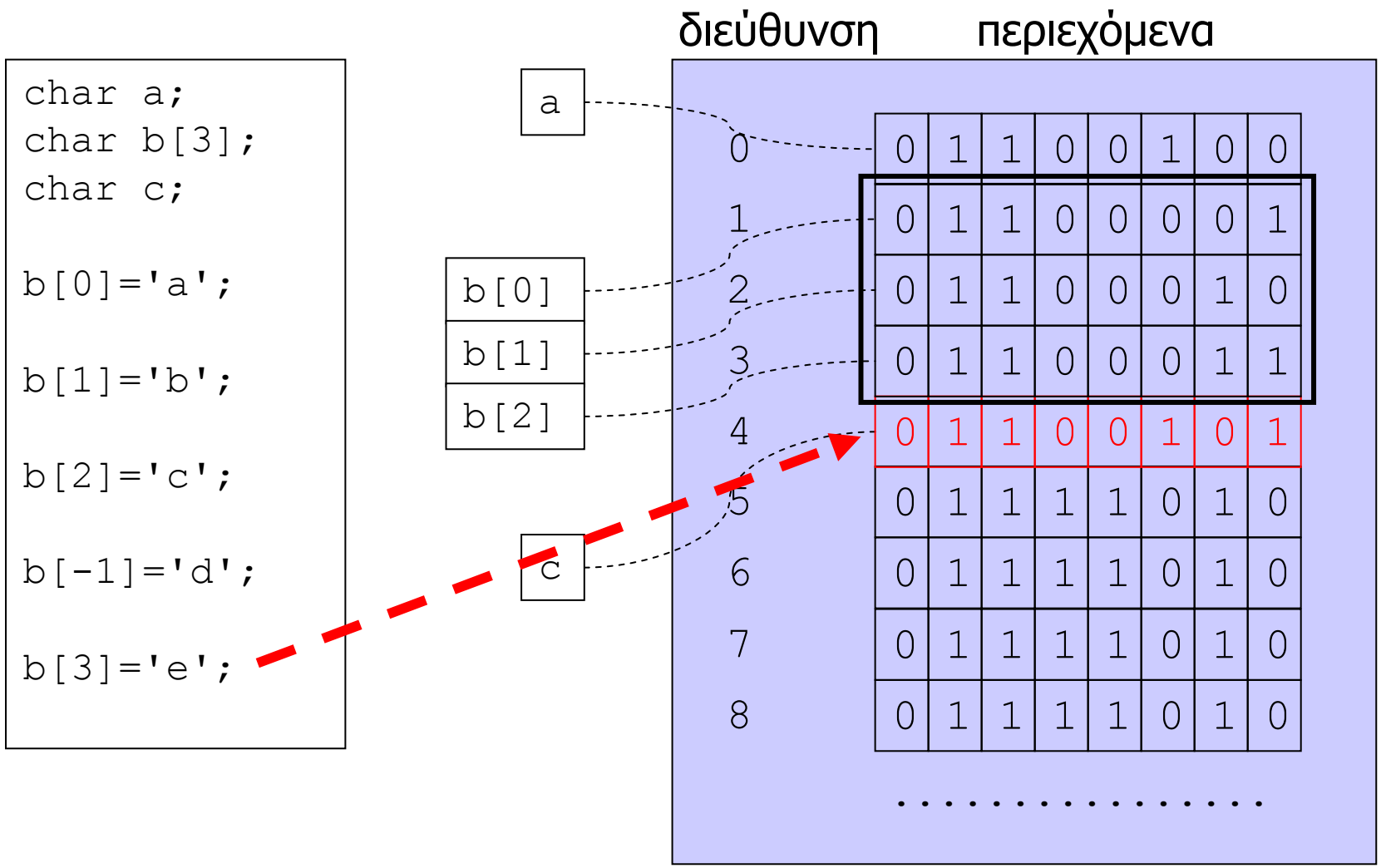
```
char a;
char b[3];
char c;

b[0]='a';

b[1]='b';
```



```
char a;
char b[3];
char c;

b[0]='a';

b[1]='b';

b[2]='c';
```




διεύθυνση
περιεχόμενα
char a;
char b[3];
char c;
b[0]='a';
b[1]='b';
b[2]='c';
b[-1]='d';
a
b[0]
b[1]
b[2]
c
0 1 1 0 0 1 0 0
0 1 1 0 0 0 0 1
0 1 1 0 0 0 1 0
0 1 1 0 0 0 1 1
0 1 1 1 1 0 1 0
0 1 1 1 1 0 1 0
0 1 1 1 1 0 1 0
0 1 1 1 1 0 1 0
0
1
2
3
4
5
6
7
8


διεύθυνση
περιεχόμενα
char a;
char b[3];
char c;
b[0]='a';
b[1]='b';
b[2]='c';
b[-1]='d';
b[3]='e';
a
b[0]
b[1]
b[2]
c
0 1 1 0 0 1 0 0
0 1 1 0 0 0 0 1
0 1 1 0 0 0 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 1 0 1
0 1 1 1 1 0 1 0
0 1 1 1 1 0 1 0
0 1 1 1 1 0 1 0
0 1 1 1 1 0 1 0
0
1
2
3
4
5
6
7
8

Πρόσβαση μέσω μεταβλητών

- Ο προσδιορισμός της θέσης των στοιχείου ενός πίνακα μπορεί να γίνει μέσω **οποιασδήποτε έκφρασης** αποτιμάται σε ακέραια τιμή.
- Αντί για σταθερές και κυριολεκτικά, μπορεί να χρησιμοποιούνται τιμές από μεταβλητές ακεραίων.
- Έτσι μπορεί να παραμετροποιηθεί η πρόσβαση στα στοιχεία ενός πίνακα `a` μεγέθους `N` μέσω ενός απλού σχήματος επανάληψης:
  - χρησιμοποιούμε μια μεταβλητή «μετρητή» `i`, που λαμβάνει τιμές από `0` μέχρι `N-1`
  - στο σώμα της επανάληψης αναφερόμαστε στο «επόμενο» στοιχείο του πίνακα ως `a[i]`

```c
/* ανάγνωση 10 ακεραίων και εκτύπωση αθροίσματος */
#include <stdio.h>
#define N 10 /* όπου "N" αντικατέστησε με "10" */

int main(int argc, char* argv[]) {
  int v[N];  /* πίνακας N ακεραίων */
  int s;     /* άθροισμα στοιχείων */
  int i;     /* μετρητής βρόγχου for */

  for (i=0; i<N; i++) {
    printf("enter int: ");
    scanf("%d", &v[i]);
  }

  s = 0;
  for (i=0; i<N; i++) {
    s = s + v[i];
  }

  printf("sum is %d\n", s);
}
```

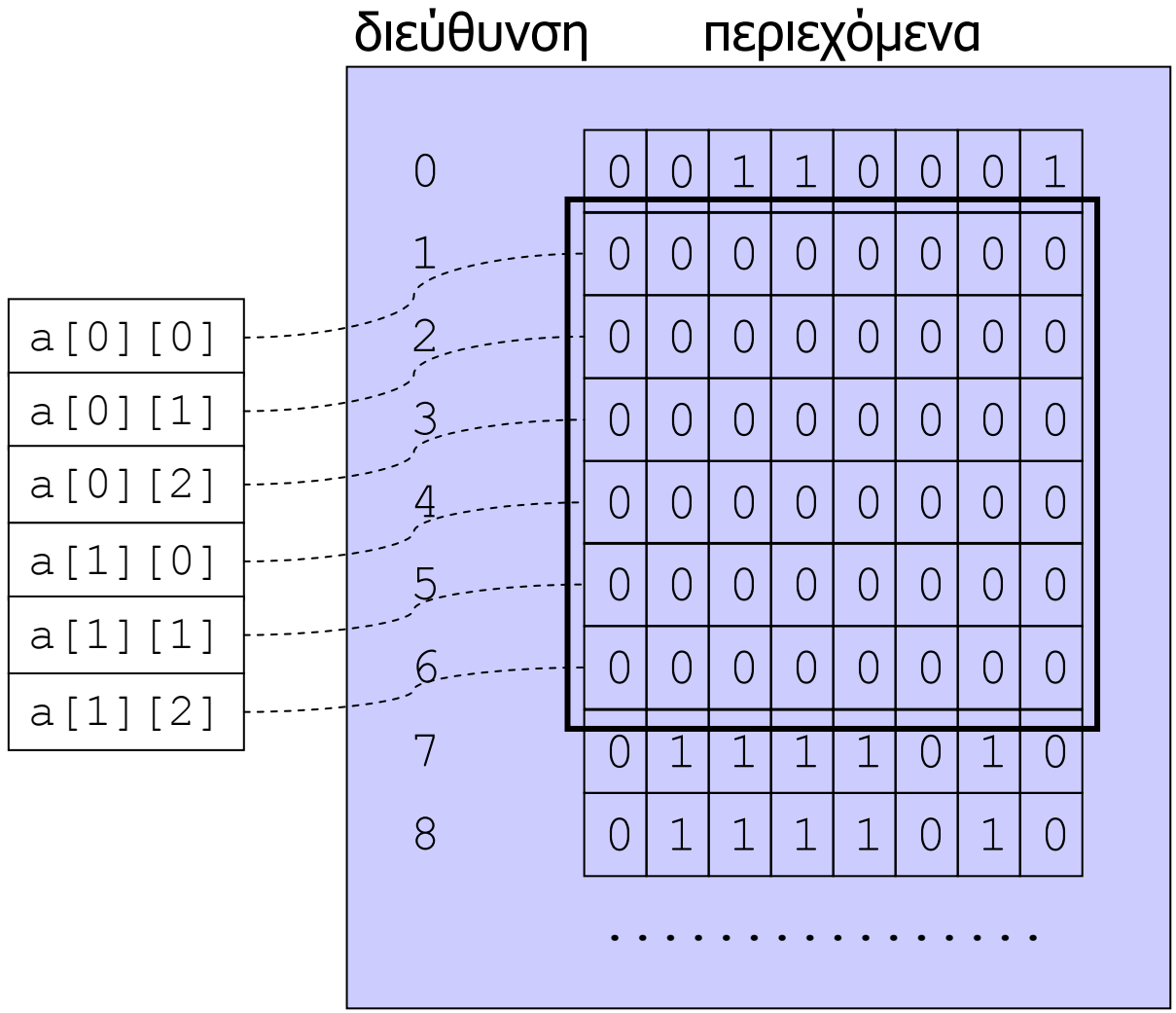
```c
/* ανάγνωση 10 ακεραίων και εκτύπωση μέγιστης τιμής */
#include <stdio.h>
#define N 10 /* όπου "N" αντικατέστησε με "10" */

int main(int argc, char* argv[]) {
  int v[N];  /* πίνακας N ακεραίων */
  int maxp;  /* θέση μεγαλύτερου στοιχείου */
  int i;     /* μετρητής βρόγχου for */

  for (i=0; i<N; i++) {
    printf("enter int: ");
    scanf("%d", &v[i]);
  }

  maxp = 0;
  for (i=1; i<N; i++) {
    if (v[maxp] < v[i]) { maxp = i; }
  }

  printf("maximum is %d\n", v[maxp]);
}
```

```
/* ανάποδη εκτύπωση εισόδου – ως 10 χαρακτήρες ή '\n' */
#include <stdio.h>
#define N 80

int main(int argc, char *argv[]) {
  char buf[N];  /* πίνακας χαρακτήρων */
  int n;        /* θέση τελευταίου έγκυρου στοιχείου */
  char c;       /* βοηθητική μεταβλητή */

  n=0;
  do {
    c = getchar();
    buf[n] = c;
    n++;
  } while ((n < N) && (c != '\n'));

  n--;
  while (n >= 0) {
    putchar(buf[n]);
    n--;
  }
  putchar('\n');
}
```

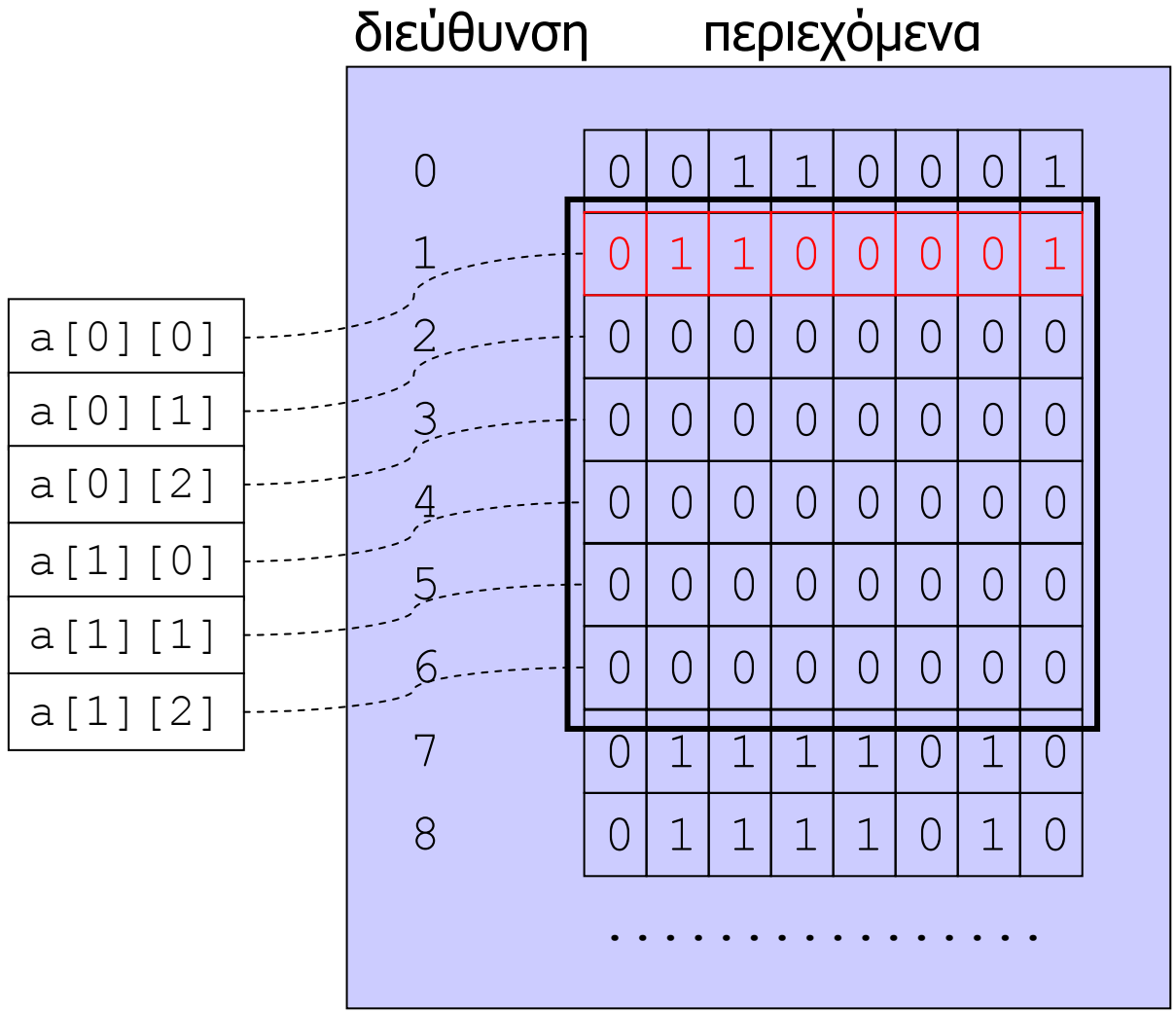
Πίνακες πολλών διαστάσεων

- Ένας πίνακας μπορεί να δηλωθεί ως Ν-διάστατος, π.χ. για 2 διαστάσεις `a[5][10]`.
- Η προσπέλαση στα στοιχεία του πίνακα γίνεται με αντίστοιχο τρόπο, π.χ. `a[0][0]` ή `a[4][9]`.
- Όπως και στους μονοδιάστατους πίνακες, η μνήμη δεσμεύεται συνεχόμενα για όλα τα στοιχεία.
- Η αποθήκευση γίνεται «σε σειρές», δηλαδή πρώτα αποθηκεύονται τα στοιχεία ως προς την τελευταία διάσταση, μετά ως προς την προ-τελευταία κλπ.
- Ένας πίνακας `a[N][M]` μπορεί να «υλοποιηθεί» εξίσου μέσω ενός πίνακα `b[N*M]` όπου αντί για `a[i][j]` γράφουμε `b[i*M + j]`.

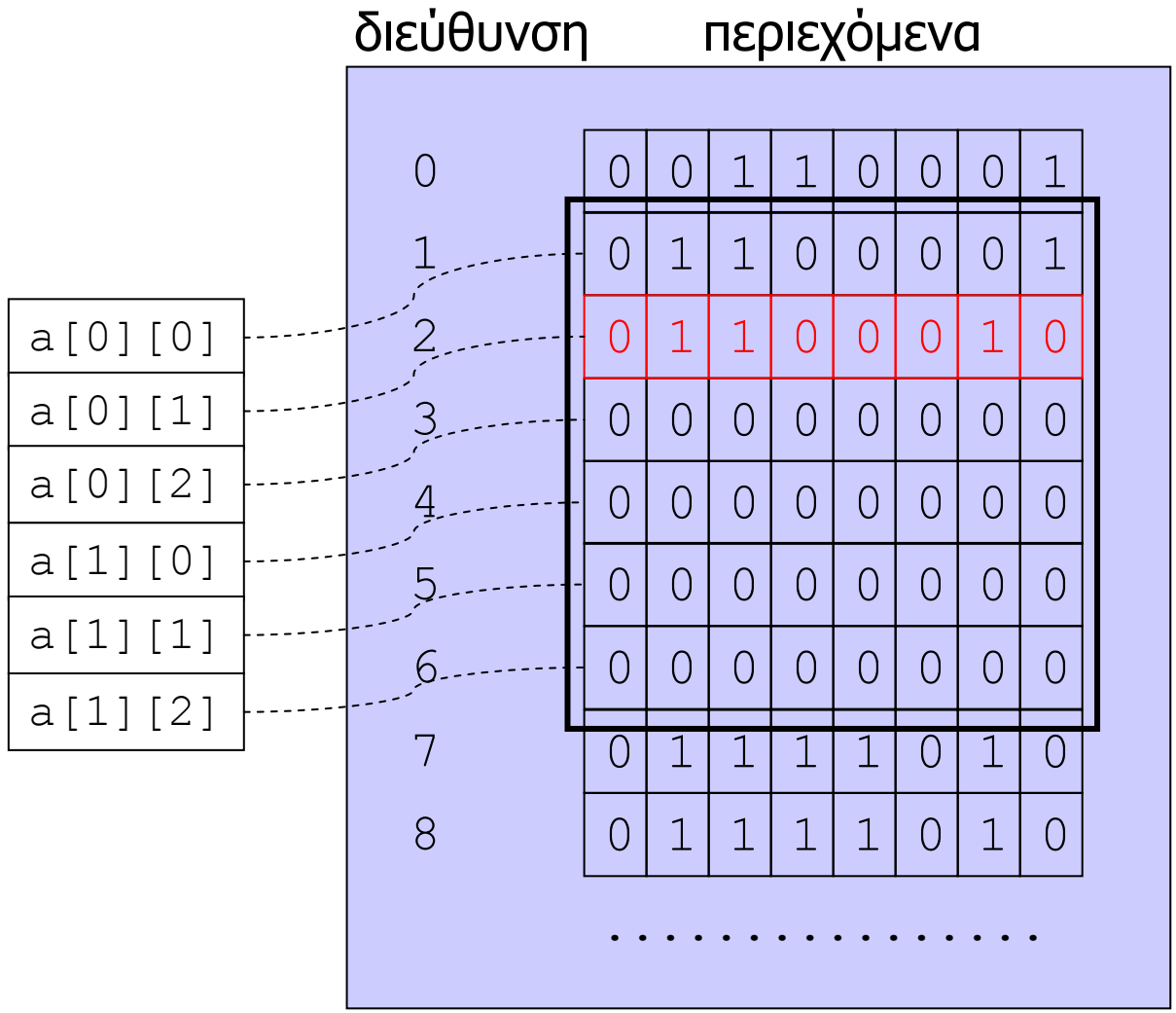
```
…
char a[2][3];
```


διεύθυνση
περιεχόμενα
a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]
0 0 0 1 1 0 0 0 1
1 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0
3 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 0 0
6 0 0 0 0 0 0 0 0
7 0 1 1 1 1 0 1 0
8 0 1 1 1 1 0 1 0
. . . . . .

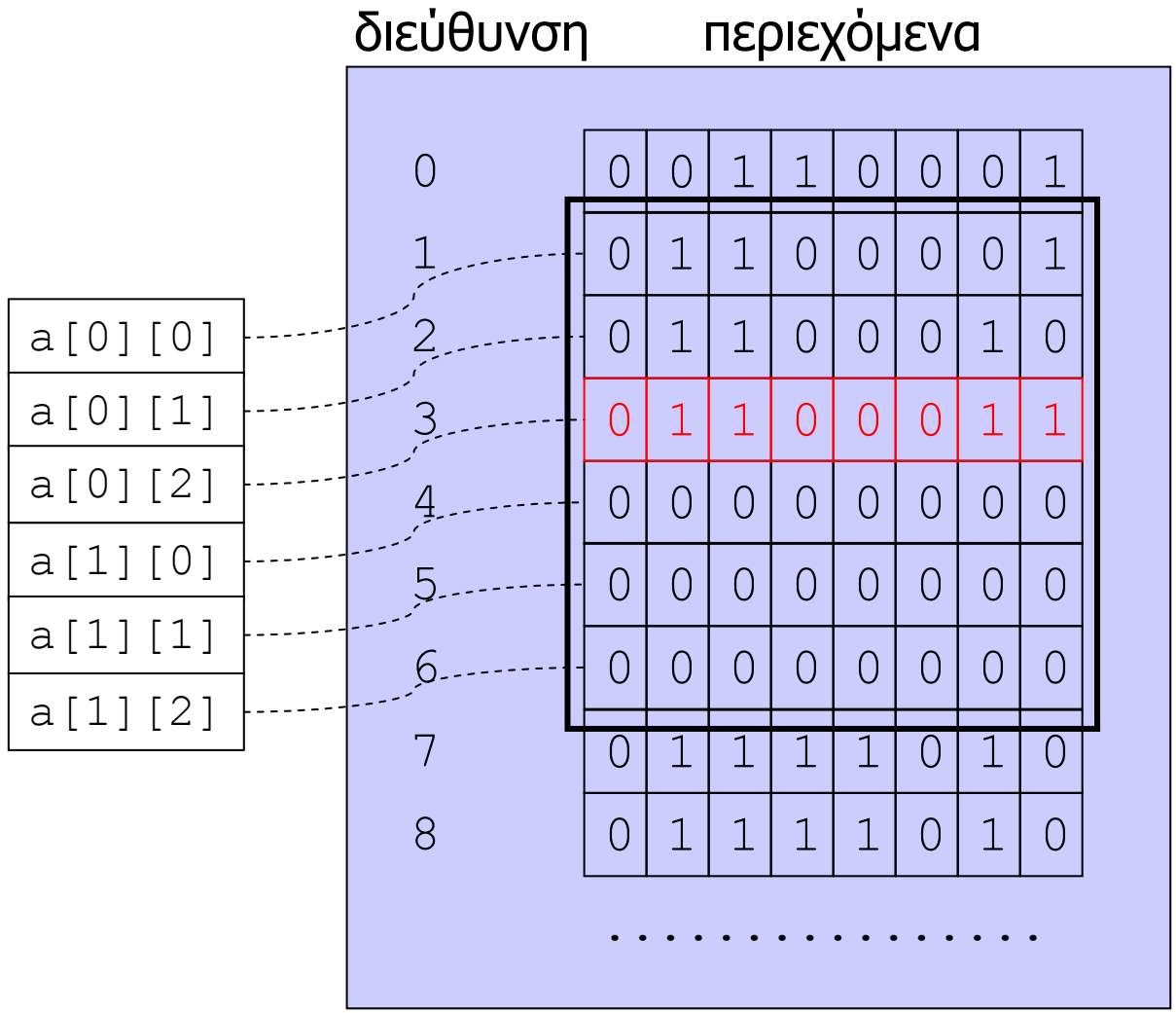
```
…
char a[2][2];

a[0][0]='a';
```



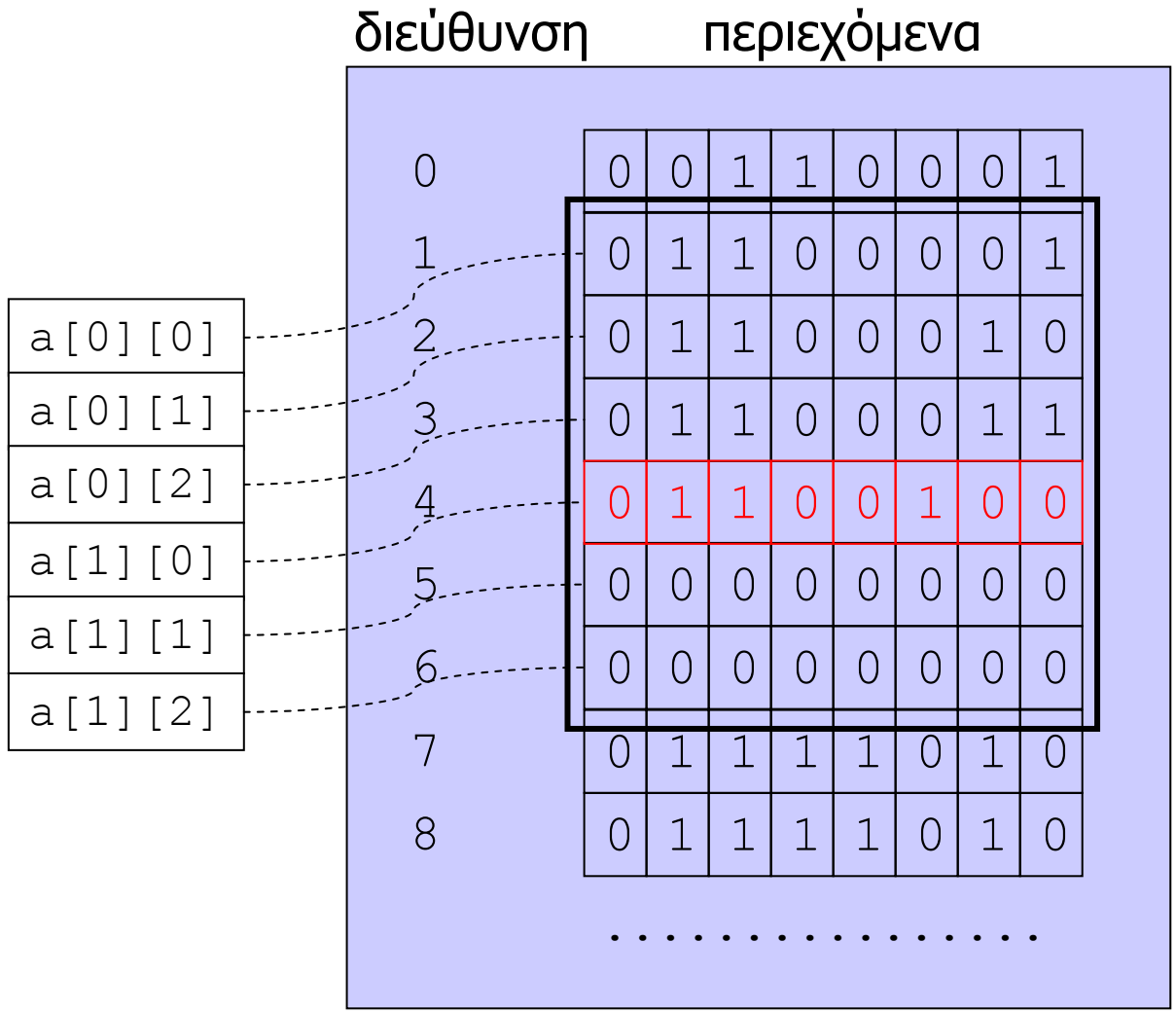
```
…
char a[2][2];

a[0][0]='a';

a[0][1]='b';
```



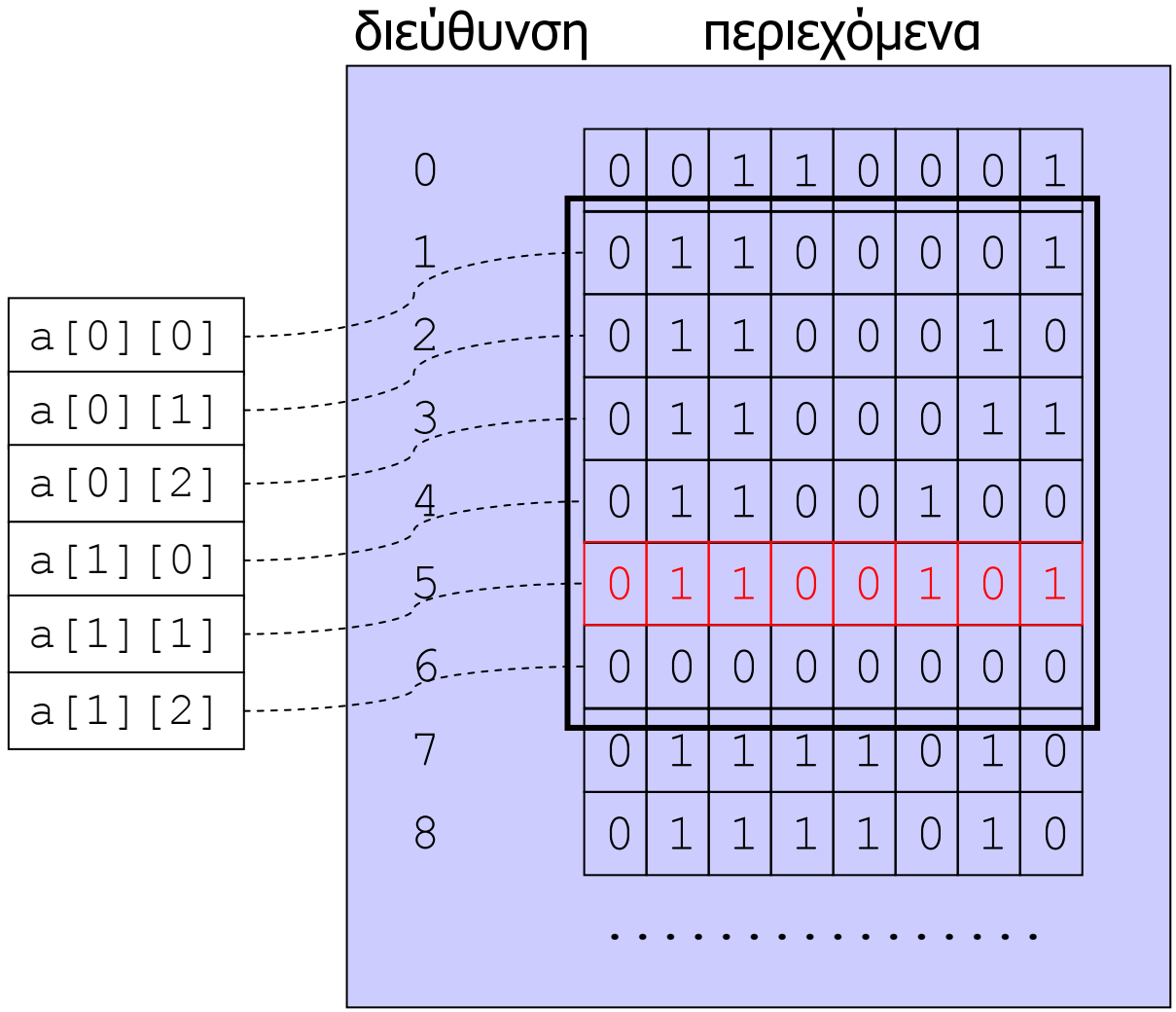
```
…
char a[2][2];

a[0][0]='a';

a[0][1]='b';

a[0][2]='c';
```



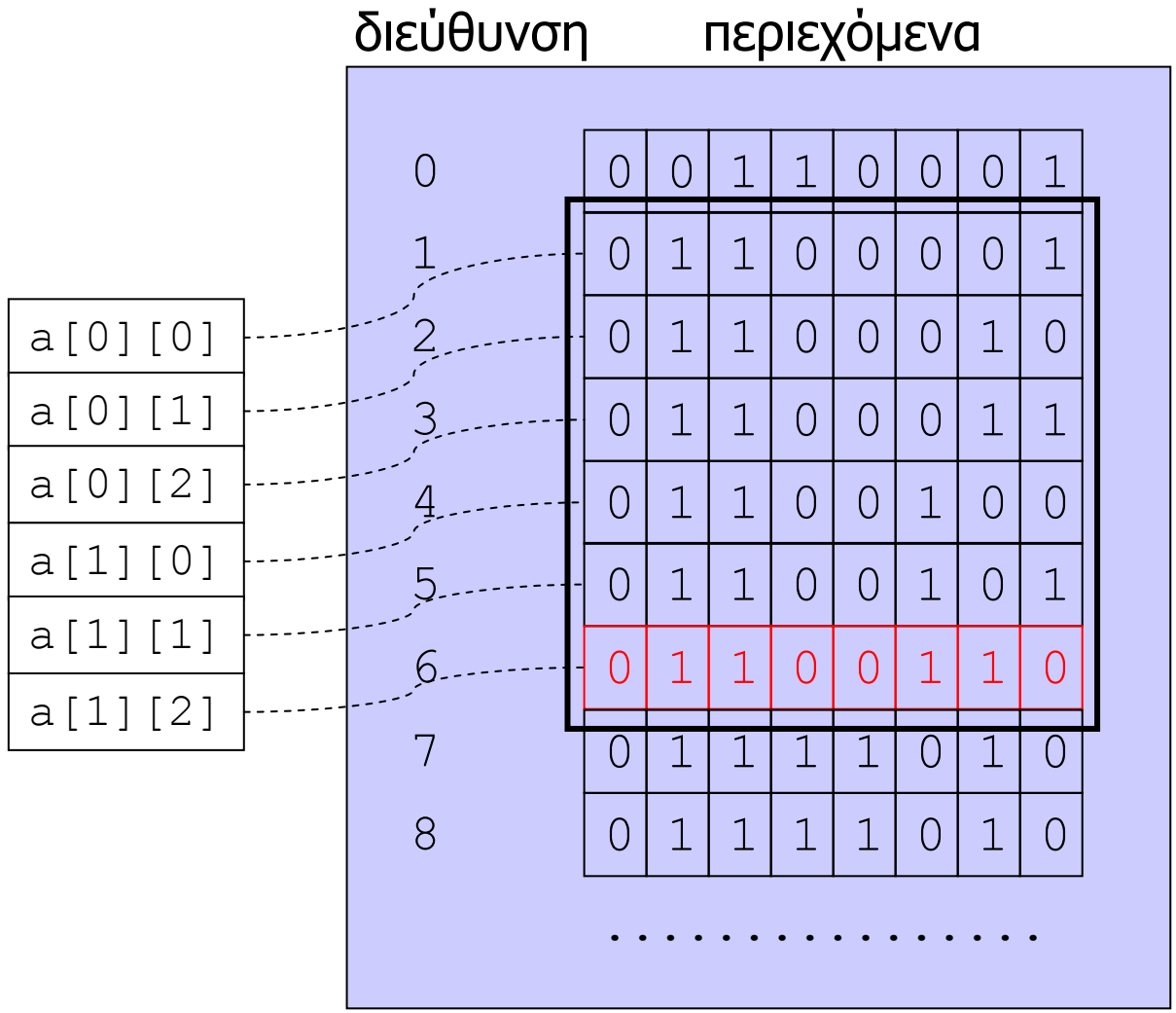
```
…
char a[2][2];

a[0][0]='a';

a[0][1]='b';

a[0][2]='c';

a[1][0]='d';
```



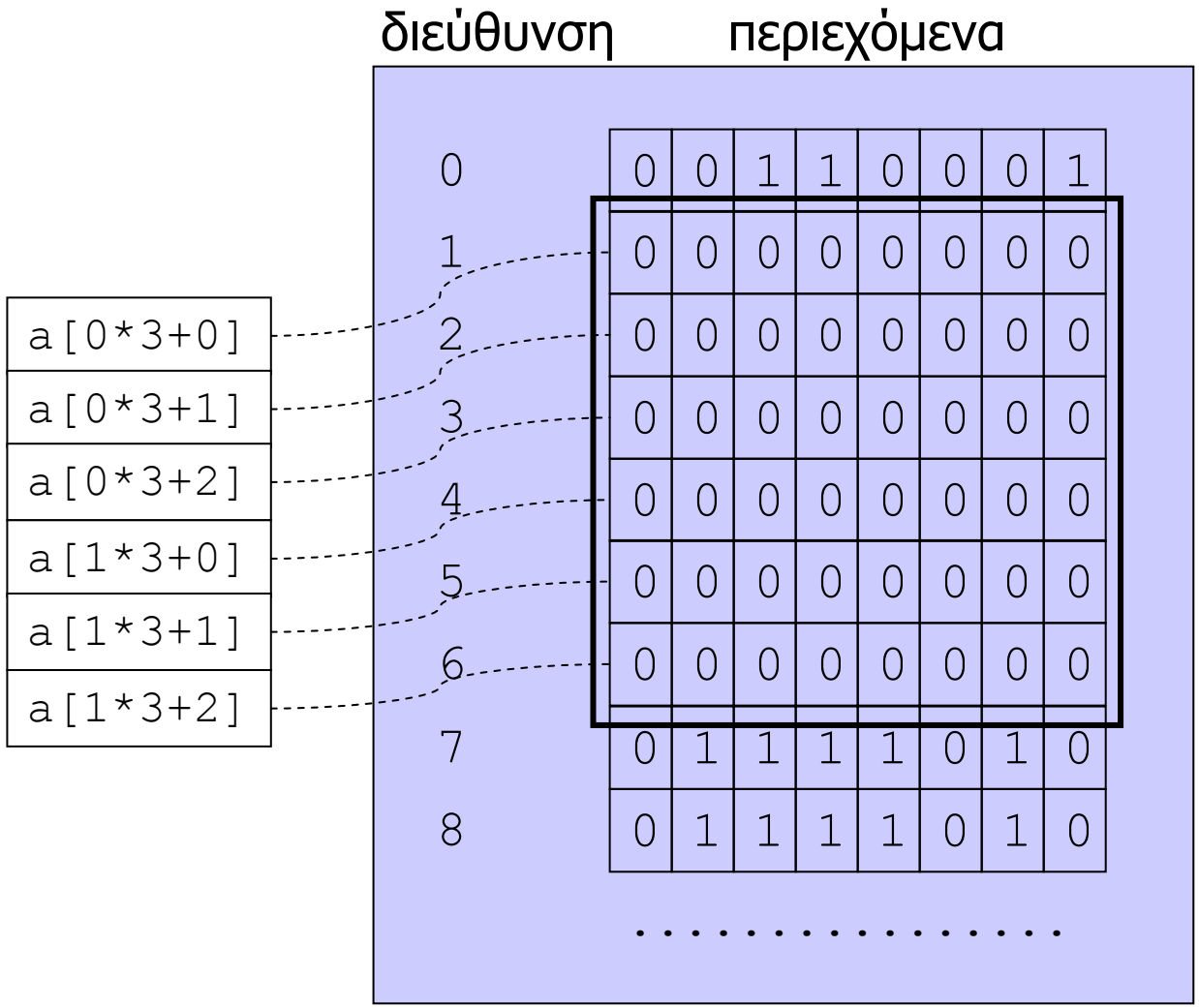
```
…
char a[2][2];

a[0][0]='a';

a[0][1]='b';

a[0][2]='c';

a[1][0]='d';

a[1][1]='e';
```



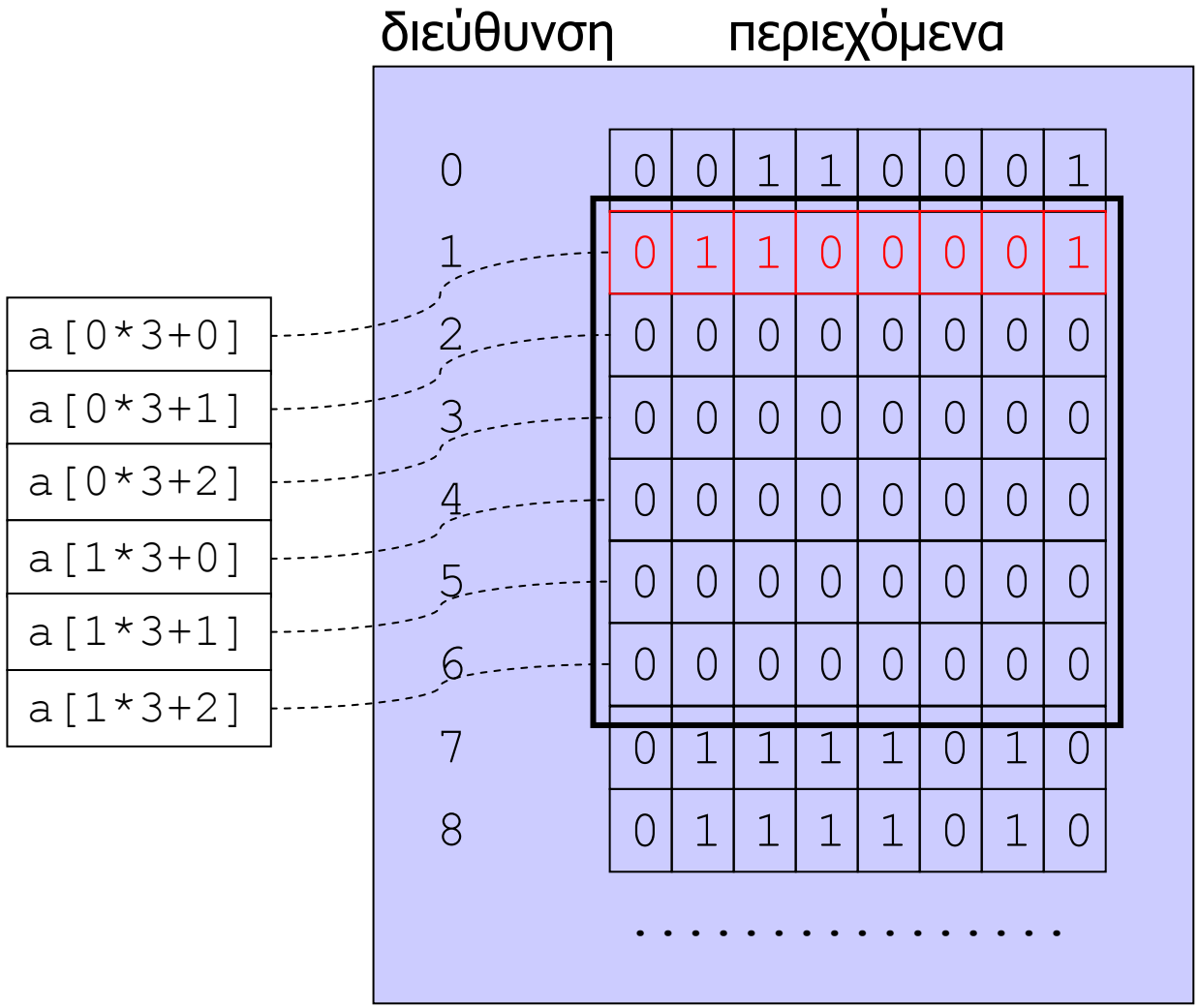
```
…
char a[2][2];

a[0][0]='a';

a[0][1]='b';

a[0][2]='c';

a[1][0]='d';

a[1][1]='e';

a[1][2]='f';
```



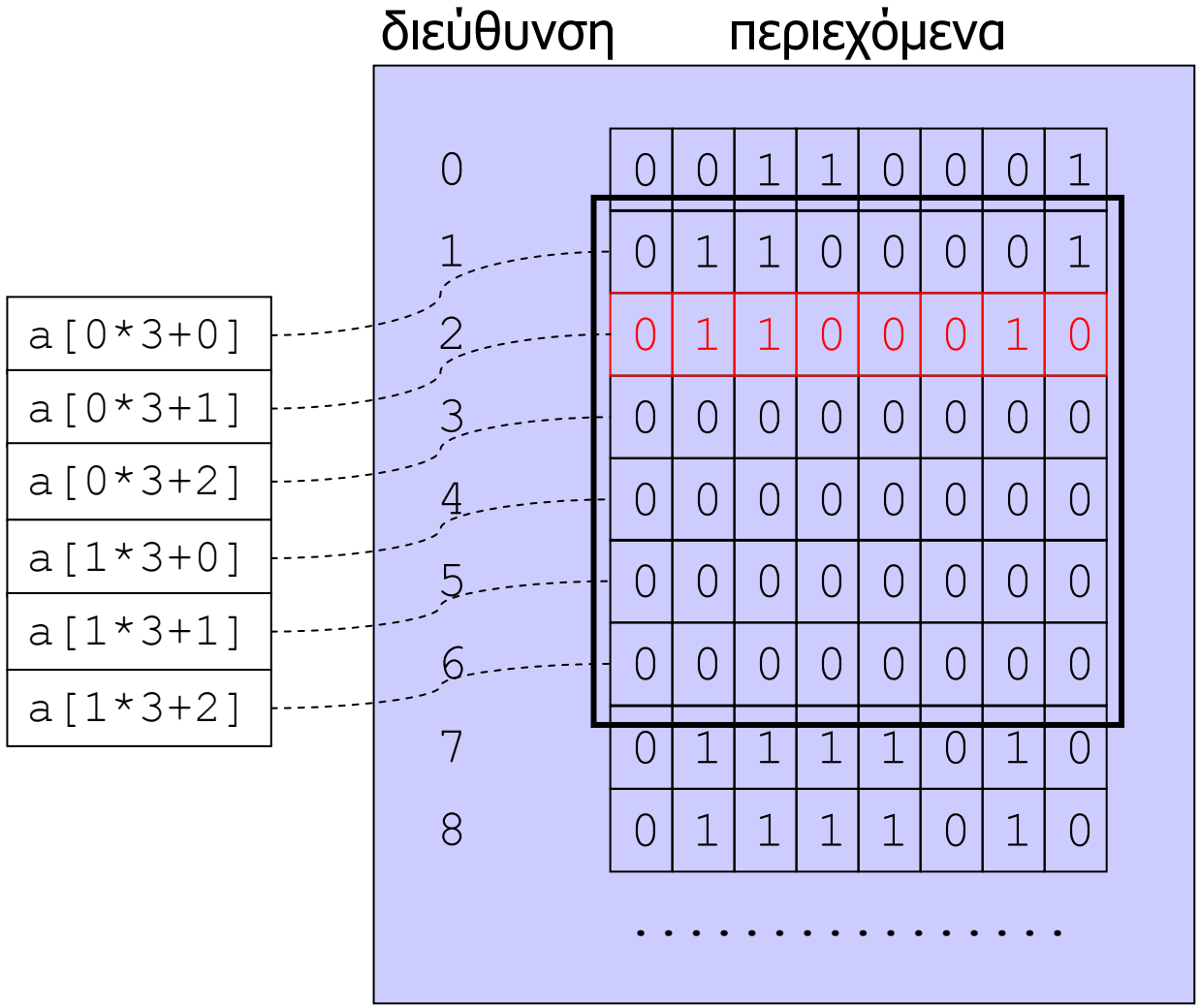
```
…
char a[2*3];
```


διεύθυνση
περιεχόμενα
a[0*3+0]
a[0*3+1]
a[0*3+2]
a[1*3+0]
a[1*3+1]
a[1*3+2]
0
1
2
3
4
5
6
7
8
0 0 1 1 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 1 1 0 1 0
0 1 1 1 1 0 1 0
. . . . . .

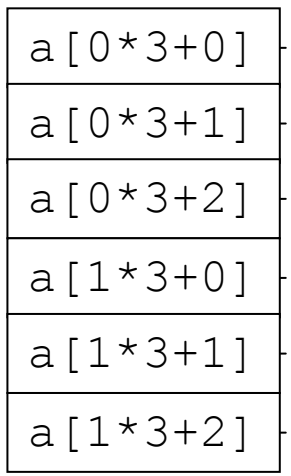
```
…
char a[2*3];

a[0]='a';
```


διεύθυνση
περιεχόμενα
a[0*3+0]
a[0*3+1]
a[0*3+2]
a[1*3+0]
a[1*3+1]
a[1*3+2]
0 0 0 1 1 0 0 0 1
1 0 1 1 0 0 0 0 1
2 0 0 0 0 0 0 0 0
3 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 0 0
6 0 0 0 0 0 0 0 0
7 0 1 1 1 1 0 1 0
8 0 1 1 1 1 0 1 0
. . . . . .

```
…
char a[2*3];

a[0]='a';

a[1]='b';
```



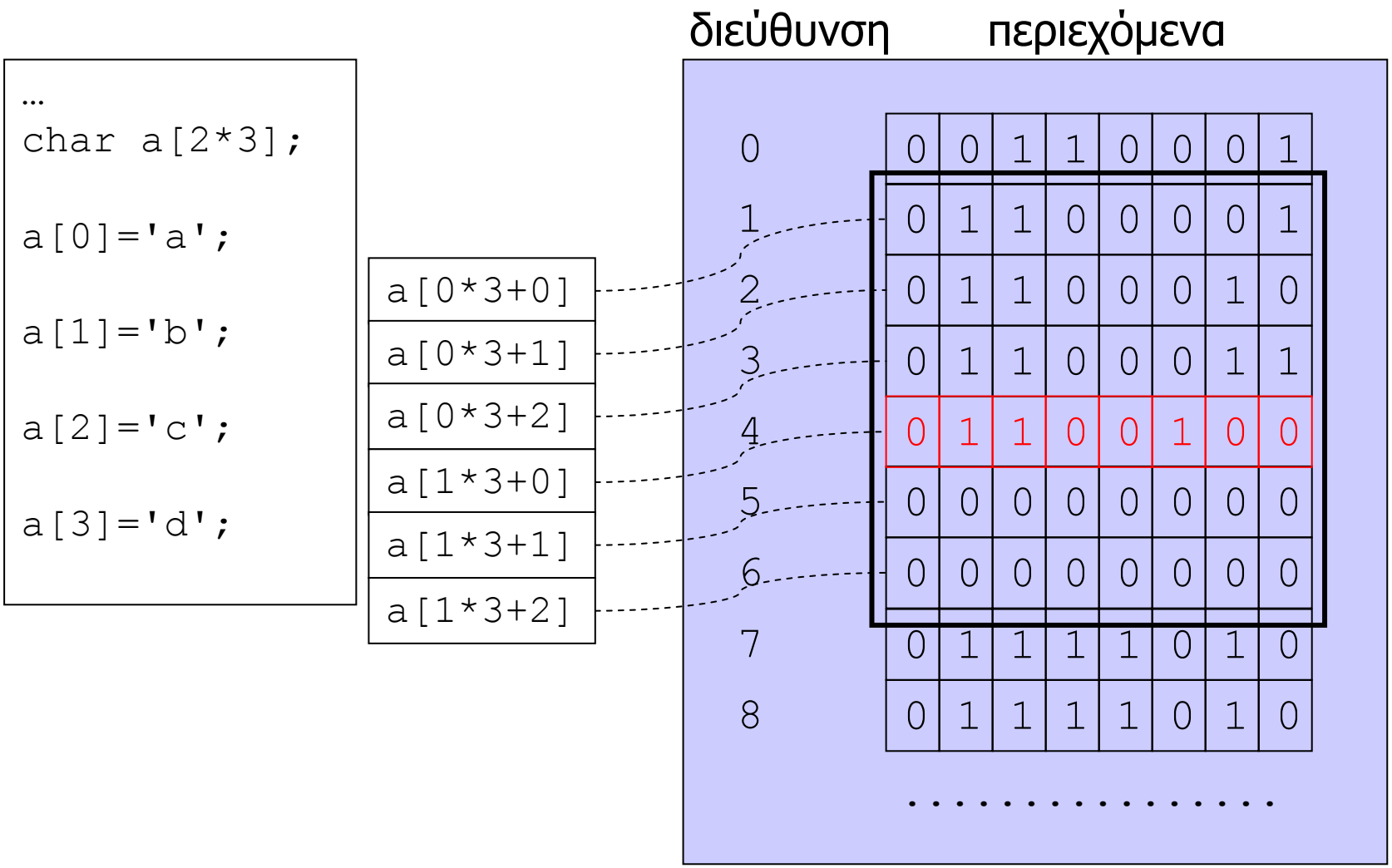
```
…
char a[2*3];

a[0]='a';

a[1]='b';

a[2]='c';
```



διεύθυνση περιεχόμενα

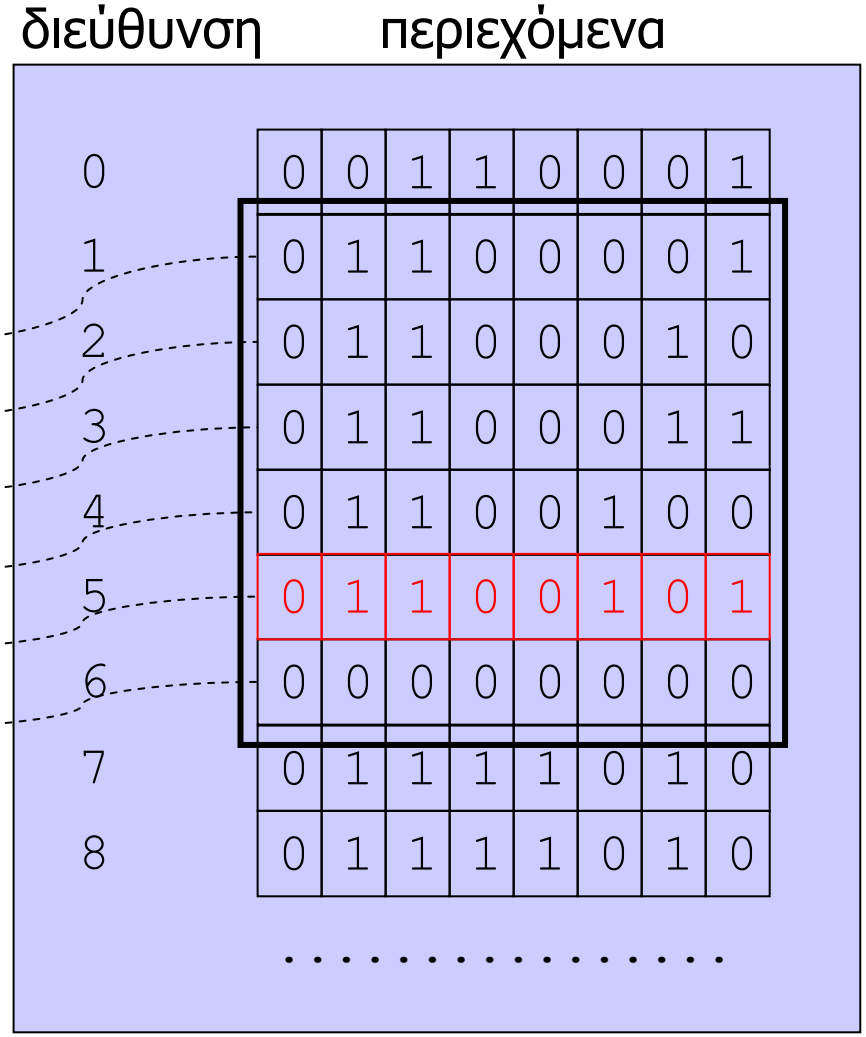
0	0	0	1	1	0	0	0	1
1	0	1	1	0	0	0	0	1
2	0	1	1	0	0	0	1	0
3	0	1	1	0	0	0	1	1
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	1	1	1	1	0	1	0
8	0	1	1	1	1	0	1	0

. . . . . .

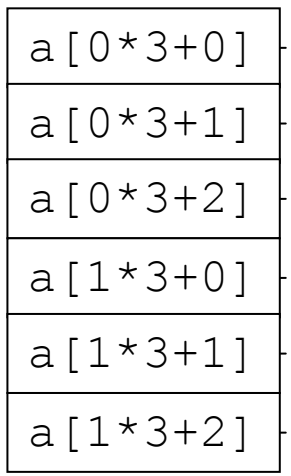

διεύθυνση
περιεχόμενα
…
char a[2*3];
a[0]='a';
a[1]='b';
a[2]='c';
a[3]='d';
a[0*3+0]
a[0*3+1]
a[0*3+2]
a[1*3+0]
a[1*3+1]
a[1*3+2]
0 0 1 1 0 0 0 1
0 1 1 0 0 0 0 1
0 1 1 0 0 0 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 1 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 1 1 0 1 0
0 1 1 1 1 0 1 0
0
1
2
3
4
5
6
7
8

```
…
char a[2*3];

a[0]='a';

a[1]='b';

a[2]='c';

a[3]='d';

a[4]='e';
```

a[0*3+0]
a[0*3+1]
a[0*3+2]
a[1*3+0]
a[1*3+1]
a[1*3+2]



```
…
char a[2*3];

a[0]='a';

a[1]='b';

a[2]='c';

a[3]='d';

a[4]='e';

a[5]='f';
```



διεύθυνση περιεχόμενα

0	0	0	1	1	0	0	0	1
1	0	1	1	0	0	0	0	1
2	0	1	1	0	0	0	1	0
3	0	1	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	0	0	1	0	1
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	1	0
8	0	1	1	1	1	0	1	0

. . . . . .

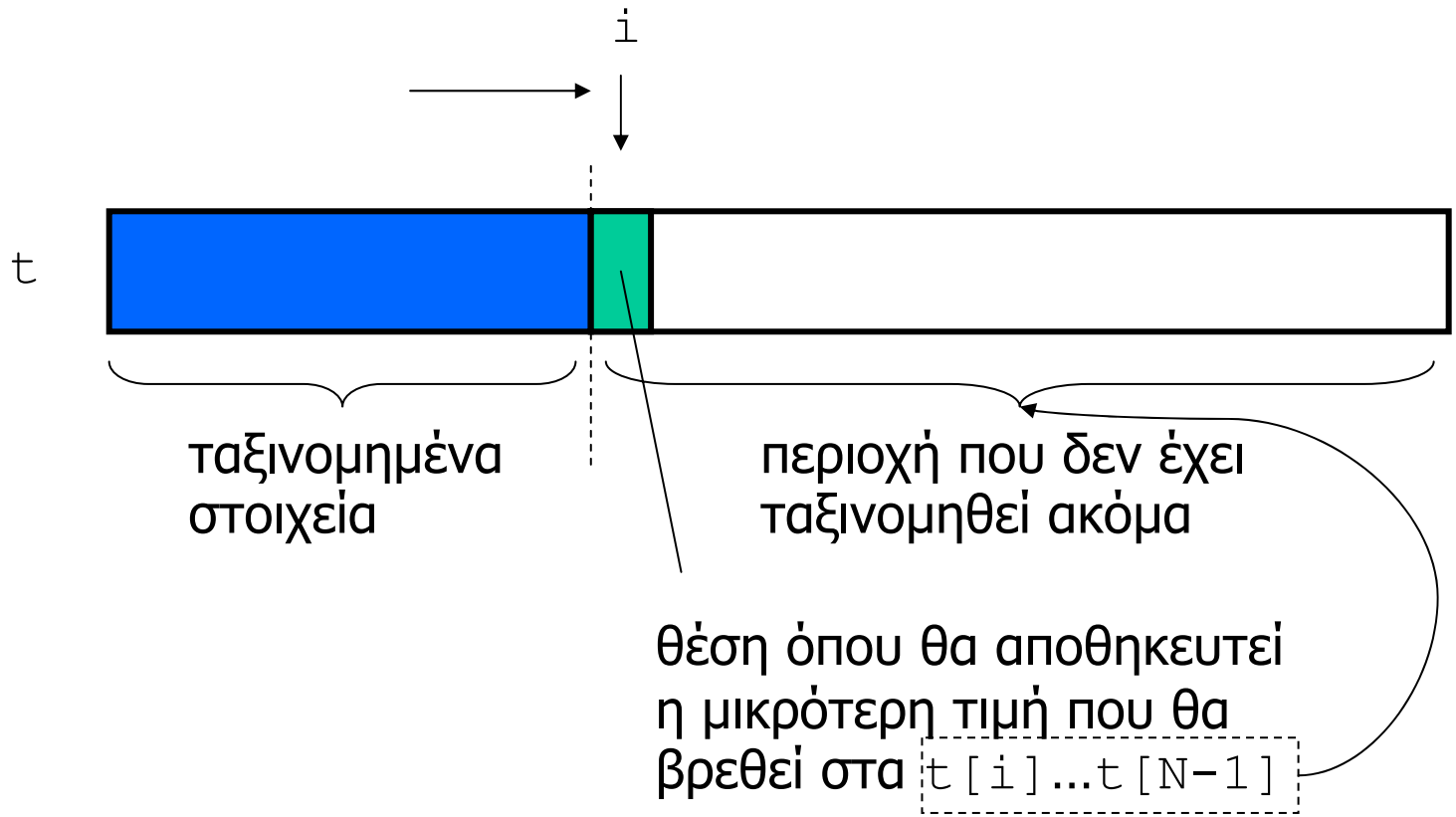
Παρένθεση
(αναζήτηση σε ταξινομημένο πίνακα)

Πρόβλημα

- Δίνεται πίνακας `t` από `N` ακεραίους.
- Ζητούμενο: να ταξινομηθούν τα περιεχόμενα του πίνακα σε αύξουσα αριθμητική σειρά:

```
∀i : 0<=i<N-1 => t[i] <= t[i+1]
```

- Μια λύση:
- Αρχίζοντας από το 1ο στοιχείο και φτάνοντας μέχρι το Νο στοιχείο του πίνακα, βρίσκουμε το στοιχείο με την μικρότερη τιμή και ανταλλάσσουμε την τιμή του με την τιμή του 1ου στοιχείου.
- Επαναλαμβάνουμε τη διαδικασία, αρχίζοντας από το 2ο στοιχείο, μετά αρχίζοντας από το 3ο στοιχείο κλπ μέχρι και το Ν-1ο στοιχείο.


i
t
ταξινομημένα στοιχεία
περιοχή που δεν έχει ταξινομηθεί ακόμα
θέση όπου θα αποθηκευτεί η μικρότερη τιμή που θα βρεθεί στα t[i]...t[N-1]

```
/* ταξινόμηση πίνακα */

int t[N];  /* πίνακας με N ακεραίους */
int i,j;   /* μετρητές βρόγχων */
int k;     /* θέση τοποθέτησης μικρότερου στοιχείου */
int tmp;   /* βοηθητική μεταβλητή για ανταλλαγή θέσης */

…      /* αρχικοποίηση πίνακα t με στοιχεία */

for (i=0; i<N; i++) {
  k = i;
  for (j=i+1; j<N; j++) {
    if (t[j] < t[k]) { k = j; }
  }
  if (k != i) {
    tmp = t[i]; t[i] = t[k]; t[k] = tmp;
  }
}
```

Πρόβλημα

- Δίνεται πίνακας `t` από `N` ακεραίους, τα περιεχόμενα του οποίου είναι **ταξινομημένα**.

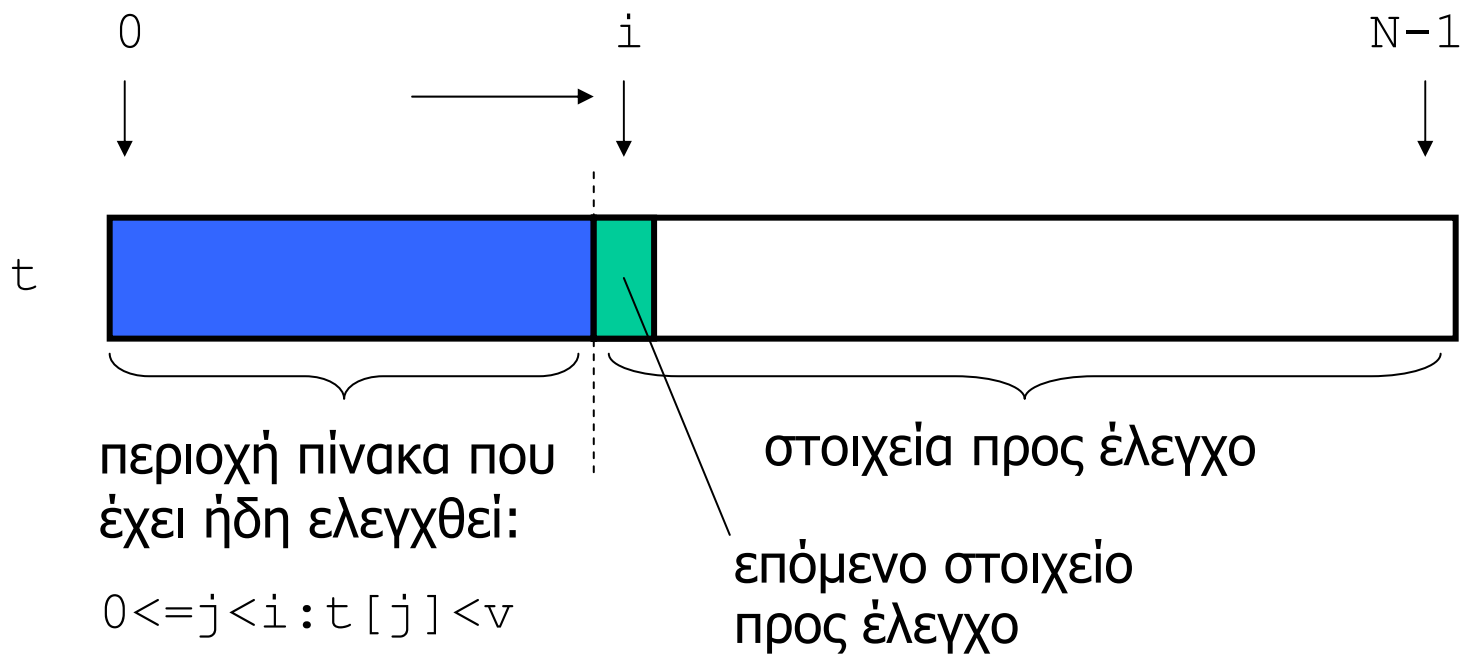
- Ζητούμενο: για μια τιμή `v`, να βρεθεί η τιμή `i`:

```
i!=N => t[i]==v
i=N  => ∀j : 0<=j<N : t[j]!=v
```

- Εξετάζουμε δύο «κλασικές» λύσεις
  - σειριακή αναζήτηση
  - δυαδική αναζήτηση (με διχοτόμηση)

Σειριακή αναζήτηση

- Αρχίζουμε από το πρώτο στοιχείο του πίνακα `t[i]`, `i=0`, που έχει (εξ' ορισμού) την μικρότερη τιμή.
- Αν `i==N`, δηλαδή ο πίνακας δεν έχει στοιχείο `t[i]`, τότε τελειώσαμε (δεν υπάρχει στοιχείο με τιμή `v`).
- Διαφορετικά, αν `t[i]==v`, τότε τελειώσαμε (αφού βρήκαμε το στοιχείο με τιμή `v`).
- Διαφορετικά, αν `t[i]>v`, τότε πάλι τελειώσαμε (είναι σίγουρο ότι δεν υπάρχει στοιχείο με τιμή `v`).
- Διαφορετικά, αν `t[i]<v`, τότε επαναλαμβάνουμε την διαδικασία για το επόμενο στοιχείο (`i=i+1`).
- Χρειάζονται κατά μέσο όρο «τάξη μεγέθους» N/2 βήματα σύγκρισης.


0
i
N-1
t
περιοχή πίνακα που
έχει ήδη ελεγχθεί:
0<=j<i:t[j]<v
στοιχεία προς έλεγχο
επόμενο στοιχείο
προς έλεγχο

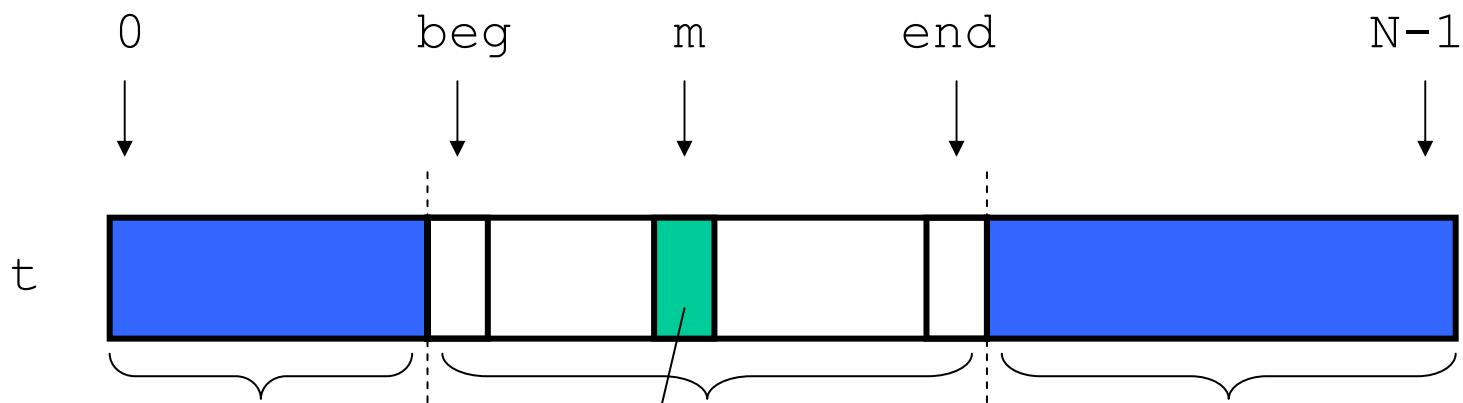
```
/* σειριακή αναζήτηση σε ταξινομημένο πίνακα */

int t[N]; /* πίνακας με N ακεραίους */
int i;    /* μετρητής βρόγχου */
int v;    /* τιμή στοιχείου που αναζητούμε */

...    /* αρχικοποίηση & ταξινόμηση πίνακα t */

for (i=0; i<N; i++) {
  if (t[i] >= v) { break; }
}

if (i == N) { printf("not found\n"); }
else if (t[i] != v) { printf("not found\n"); }
else { printf("found at position %d\n", i); }
```

```
/* σειριακή αναζήτηση σε ταξινομημένο πίνακα */

int t[N]; /* πίνακας με N ακεραίους */
int i;    /* μεταβλητή βρόγχου */
int v;    /* τιμή στοιχείου που αναζητούμε */

...    /* αρχικοποίηση & ταξινόμηση πίνακα t */

for (i=0; (i<N) && (t[i]<v); i++) {}

if (i == N) { printf("not found\n"); }
else if (t[i] != v) { printf("not found\n"); }
else { printf("found at position %d\n", i); }
```

η σύμβαση εκτέλεσης του `&&` εγγυάται ότι **δεν** θα επιχειρηθεί πρόσβαση `t[i]` όταν το `i` φτάσει το όριο `N` του πίνακα

Δυαδική αναζήτηση

- Θέτουμε τα όρια αναζήτησης `beg=0` και `end=N-1`.
- Αρχίζουμε από το μεσαίο στοιχείο του πίνακα `t[m]`, όπου `m = (beg+end)/2`.
- Αν `t[m] == v`, τελειώσαμε.
- Αν `t[m] < v`, συνεχίζουμε με τον (μισό) υποπίνακα που έχει στοιχεία με **μεγαλύτερες** τιμές του `t[m]` δηλαδή θέτουμε `beg = m+1`.
- Αν `t[m] > v`, συνεχίζουμε με τον (μισό) υποπίνακα που έχει στοιχεία με **μικρότερες** τιμές του `t[m]` δηλαδή θέτουμε `end = m-1`.
- Αν `beg > end` τελειώσαμε (η τιμή `v` δεν υπάρχει).
- Χρειάζονται «τάξη μεγέθους» $\log_2 N$ βήματα.



περιοχή που έχει «αποκλεισθεί:

`0<=i<beg: t[i]<v`

περιοχή προς έλεγχο

επόμενο στοιχείο προς έλεγχο

περιοχή που έχει «αποκλεισθεί:

`end<=i<N: t[i]>v`

```
/* δυαδική αναζήτηση σε ταξινομημένο πίνακα */

int t[N];      /* πίνακας με N ακεραίους */
int beg,end;   /* όρια αναζήτησης [beg,end) */
int m;         /* τιμή μεσαίου στοιχείου */
int v;         /* τιμή στοιχείου που αναζητούμε */

...    /* αρχικοποίηση & ταξινόμηση πίνακα t */

beg = 0; end = N-1;
while (beg <= end) {
  m = (beg+end)/2;
  if (t[m] == v) { break; }
  else if (t[m] < v) { beg = m+1; }
  else /* (t[m] > v) */ { end = m-1; }
}

if (beg > end) { printf("not found\n"); }
else { printf("found at position %d\n", m); }
```

Σύγκριση

# στοιχείων πίνακα	μ.ο βημάτων σειριακής αναζ.	μ.ο βημάτων δυαδικής αναζ.
`N`	`N/2`	`log N`
100	50	7
1000	500	10
1000000	500000	20
1000000000	500000000	30

- Η διαφορά γίνεται **τεράστια** για μεγάλες τιμές του N.

Παρένθεση
(αναζήτηση σε ταξινομημένο πίνακα)

Συμβολοσειρές – Αλφαριθμητικά (strings)

Συμβολοσειρές / σειρές χαρακτήρων (strings)

- Τα strings υλοποιούνται ως πίνακες από `char`, αλλά με την επιπρόσθετη ιδιότητα ότι **τουλάχιστον** ένα στοιχείο είναι ίσο με την τιμή '`\0`' (ή '`\0x00`' ή `0`).
- Το πρώτο στοιχείο με την τιμή '`\0`' ορίζει το τέλος του string (όχι όμως απαραίτητα και του πίνακα χαρακτήρων, τα υπόλοιπα στοιχεία του οποίου δεν λαμβάνονται υπόψη στις διάφορες πράξεις string).
- Ένα πρόγραμμα που χρησιμοποιεί strings υποθέτει πως τερματίζονται με '`\0`' και αν δημιουργεί strings φροντίζει αυτά να τερματίζονται με '`\0`' .
- Η βιβλιοθήκη `string` περιέχει αρκετές συναρτήσεις για τις πιο συχνές πράξεις με strings ώστε να μην χρειάζεται να υλοποιηθούν από τον προγραμματιστή.

```
char str1[] = {'w', 'i', 'n'};

char str2[] = {'w', 'i', 'n', '\0'};      /* "win" */

char str3[] = {'w', 'i', 'n', '\0', 'x'}; /* "win" */

char str4[]="win";                        /* "win" */

char str5[7];

str5[0] = str1[1];   /* 'i' */
str5[1] = ' ';       /* ' ' */
str5[2] = str1[0];   /* 'w' */
str5[3] = str2[1];   /* 'i' */
str5[4] = str3[2];   /* 'n' */
str5[5] = '\0';      /* '\0' */
str5[6] = 'x';       /* 'x' */
```

Ανάγνωση και εκτύπωση strings

- Μπορούμε να τα υλοποιήσουμε (εμείς) χαρακτήρα προς χαρακτήρα μέσω της `getchar` και `putchar`.
- Ή να χρησιμοποιήσουμε την `scanf` και `printf` με τον αντίστοιχο προδιορισμό, που είναι το `"%s"`.
- Η `printf` σταματά την εκτύπωση των περιεχομένων του πίνακα όταν συναντήσει στοιχείο με τιμή `'\0'`.
- Η `scanf` σταματά την ανάγνωση χαρακτήρων (και την αποθήκευση τους στον πίνακα) όταν βρεθεί «λευκός» χαρακτήρας – **προσοχή** καθώς μπορεί να γίνει αποθήκευση **εκτός ορίων** του πίνακα!
- Αντίθετα με ότι γνωρίζουμε, στην `scanf` **δεν** δίνουμε σαν παράμετρο την **διεύθυνση** της μεταβλητής πίνακα – θα δούμε σε λίγο το γιατί ...

```c
#include<stdio.h>

int main(int argc, char *argv[]) {
  char str[4];

  printf("enter string:");
  scanf("%s",str);
  printf("you entered %s\n",str);

  printf("enter string:");
  scanf("%3s",str);
  printf("you entered %s\n",str);
}
```

σαν παράμετρος δίνεται απ' ευθείας η μεταβλητή (πίνακας) αντί (όπως «θα έπρεπε») η διεύθυνση της (δεν γράφουμε &s)

αν ο χρήστης εισάγει συμβολοσειρά με περισσότερους από 3 χαρακτήρες, θα υπάρξει πρόβλημα καθώς αυτοί θα αποθηκευτούν σε θέσεις εκτός ορίων του πίνακα ...

διαβάζει το πολύ 3 χαρακτήρες (εκτός από το '\0' που θα αποθηκευτεί ως τελευταίος χαρακτήρας της συμβολοσειράς)

```c
/* μήκος string s */

#include <stdio.h>
#define N 32

int main(int argc, char *argv[]) {
  int i;
  char str[N];

  printf("enter string:");
  scanf("%31s", str);

  for (i=0; str[i]!='\0'; i++) {}

  printf("length of %s is %d\n", str, i);
}
```

```
/* ανάποδη αντιγραφή του s1 στο s2 */

#include <stdio.h>
#define N 32

int main(int argc, char *argv[]) {
  int i,j;
  char s1[N],s2[N];

  printf("enter string:");
  scanf("%31s", s1);

  /* βρες το τέλος του s1 */
  for (i=0; s1[i]!='\0'; i++) {}

  /* αντέγραψε ανάποδα στο s2 */
  for (j=0,i--; i>=0; j++,i--) { s2[j] = s1[i]; }
  s2[j] = '\0'; /* τερμάτισε το s2 */

  printf("%s in reverse is %s\n", s1, s2);
}
```

```
/* αντιγραφή του s1 "και" του s2 στο s3 */

#include <stdio.h>
#define N 32

int main(int argc, char *argv[]) {
  int i,j; char s1[N],s2[N],s3[2*N-1];

  printf("enter string:"); scanf("%31s", s1);
  printf("enter string:"); scanf("%31s", s2);

  /* αντέγραψε το s1 στο s3 */
  for (i=0; s1[i]!='\0'; i++) { s3[i] = s1[i]; }

  /* αντέγραψε το s2 στο s3 */
  for (j=0; s2[j]!='\0'; j++,i++) { s3[i] = s2[j]; }

  s3[i]='\0'; /* τερμάτισε το s3 */

  printf("%s + %s = %s\n", s1, s2, s3);
}
```

Συναρτήσεις βιβλιοθήκης

```
#include <string.h>
char *strcat(char *s1, const char *s2);
char *strchr(const char *s, int c);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
char *strcpy(char *s1, const char *s2);
char *strdup(const char *s2);
size_t strlen(const char *s);
```

```
#include <stdlib.h>
int atoi(const char *s);
double atof(const char *s);
```

```
#include <ctype.h>
int isalpha(char c);
int isupper(char c);
int islower(char c);
int isspace(char c);
int isdigit(char c);
```