



Προγραμματισμός Ι (ECE115)

#7

πίνακες και αλφαριθμητικά (strings)

Πίνακες

- Ο πίνακας είναι μια ειδική δομή για την αποθήκευση μιας **σειράς** από στοιχεία/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
 - μέγεθος $N \Rightarrow$ ο πίνακας έχει N στοιχεία
- Κάθε στοιχείο ενός πίνακα από αντικείμενα τύπου T είναι **συντακτικά συμβατό** με μια μεταβλητή τύπου T
- Όπως μια συμβατική μεταβλητή, κάθε στοιχείο του πίνακα μπορεί να αρχικοποιηθεί με συγκεκριμένη τιμή

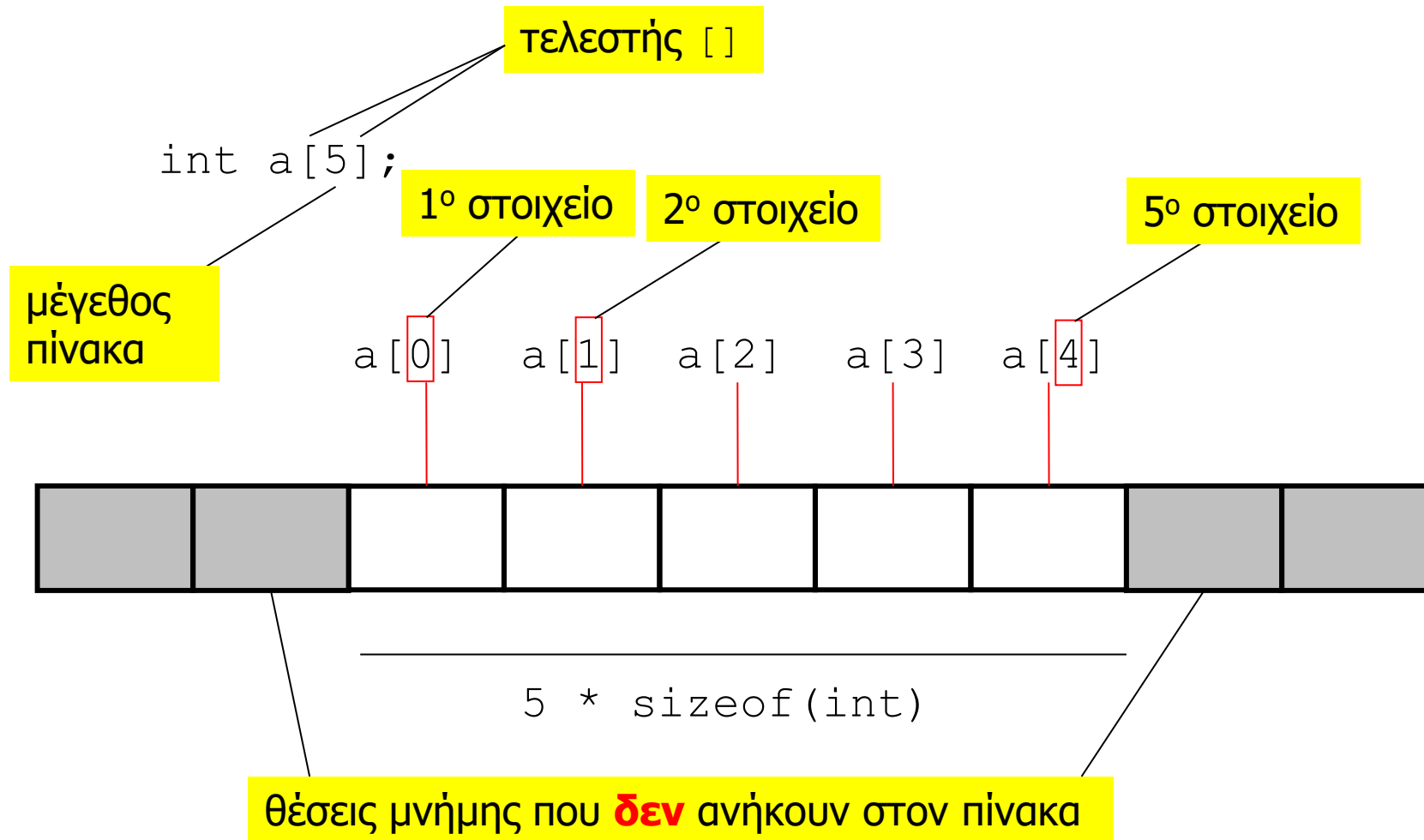
Δέσμευση μνήμης

- Ο αριθμός των στοιχείων ενός πίνακα δίνεται (άμεσα ή έμμεσα) **κατά την δήλωση** του
- Το μέγεθος του πίνακα πρέπει να δηλωθεί **την ώρα που γράφεται** ο κώδικας
 - δεν μπορεί να (επανα)προσδιοριστεί ή/και να αλλάξει κατά την διάρκεια της εκτέλεσης
- Μνήμη πίνακα μεγέθους N με στοιχεία τύπου T
 - $N * \text{sizeof}(T)$
 - δεσμεύεται **μονομιάς** ανεξάρτητα με το πόσα/ποια στοιχεία του πίνακα θα χρησιμοποιηθούν τελικά από το πρόγραμμα
- Η μνήμη των επιμέρους στοιχείων του πίνακα δεσμεύεται **συνεχόμενα**
 - η μνήμη του στοιχείου στη θέση i βρίσκεται στην **αμέσως επόμενη** διεύθυνση από αυτή του στοιχείου $i - 1$

Προσπέλαση

- Αν ο πίνακας έχει μέγεθος N , τότε η θέση **0** αντιστοιχεί στο **πρώτο** στοιχείο και η θέση **$N-1$** στο **τελευταίο** στοιχείο του πίνακα
- Η προσπέλαση θέσεων **εκτός ορίων** του πίνακα αποτελεί προγραμματιστικό λάθος
 - πρόσβαση μνήμης που δεν ανήκει στον πίνακα
- Ο μεταφραστής **δεν ελέγχει** αν η θέση που δίνει το πρόγραμμα είναι εντός των ορίων





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

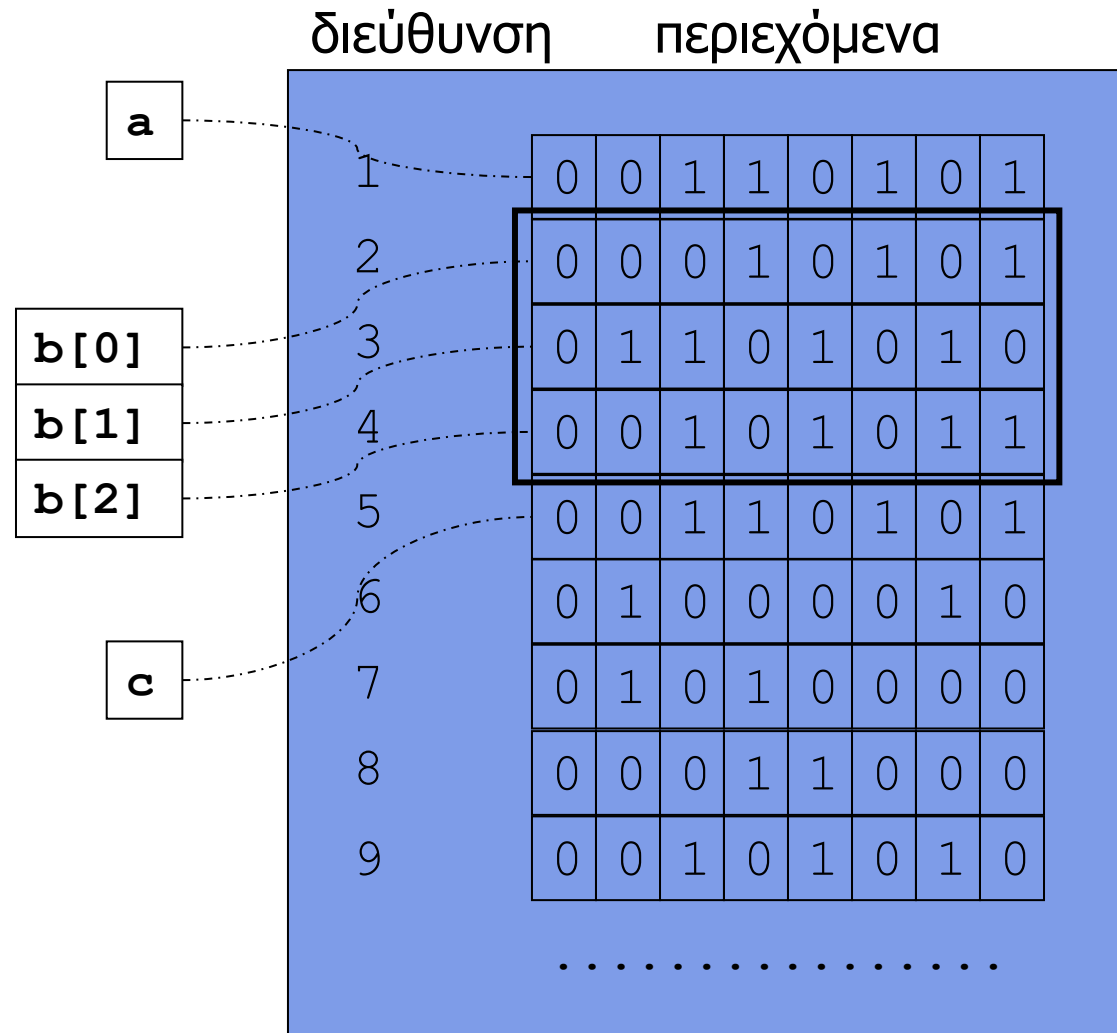
a = c[-1];            /* λάθος/άκυρη πρόσβαση! */

b[3] = 5;             /* λάθος/άκυρη πρόσβαση! */
```

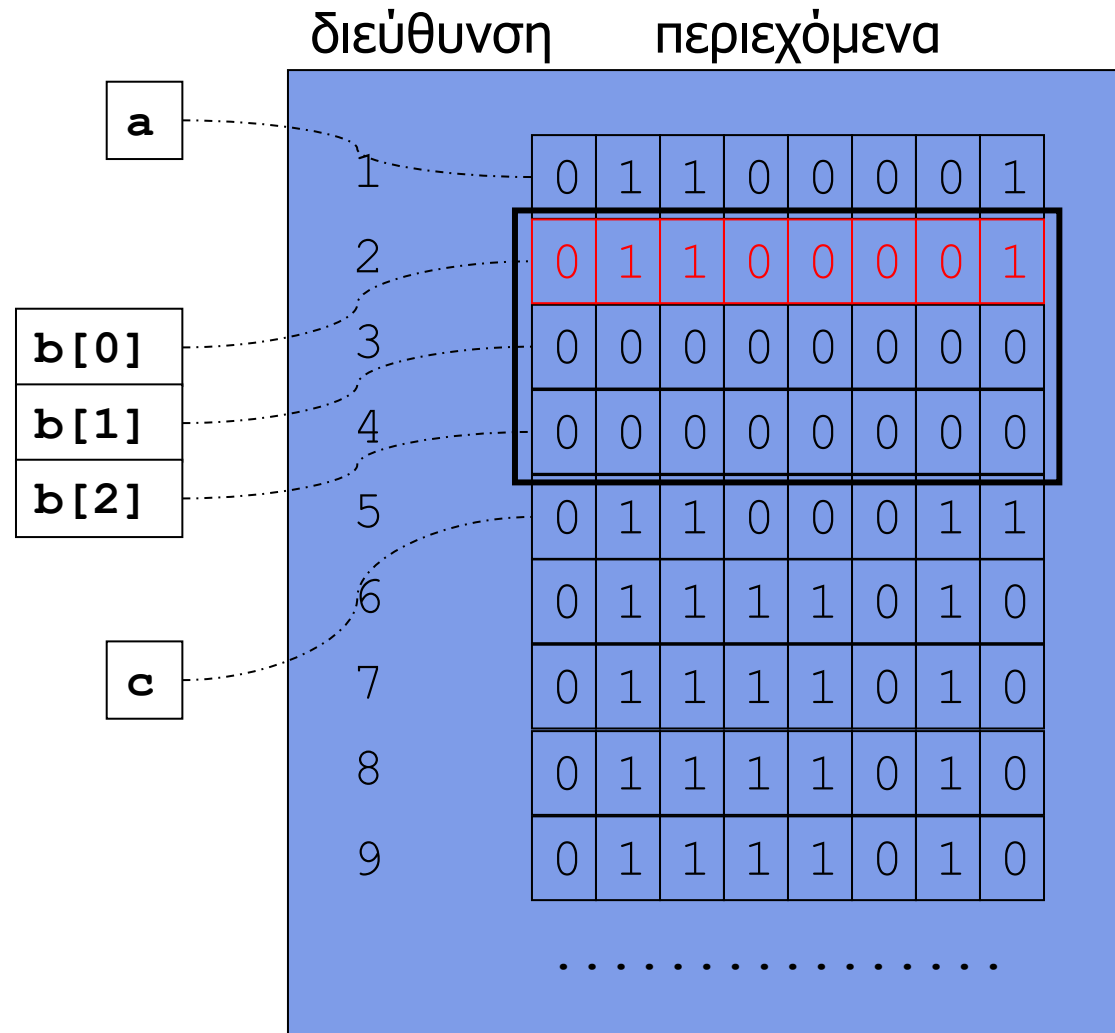
Άκυρη προσπάθειας μνήμης

- Όσο χρησιμοποιούμε απλές μεταβλητές, είναι αδύνατο να προσπελάσουμε άκυρες θέσεις μνήμης
- Αυτό **δεν** ισχύει όταν χρησιμοποιούμε πίνακες
- Η πρόσβαση σε θέση εκτός των ορίων του πίνακα **δεν εντοπίζεται** απαραίτητα από τον μεταφραστή
- **ούτε** (απαραίτητα) κατά την εκτέλεση
- Το πρόγραμμα μπορεί να «καταστρέψει» δεδομένα που αντιστοιχούν σε άλλες (δικές του) μεταβλητές
 - μπορεί να προκληθεί εντελώς **απροσδόκητη** συμπεριφορά του προγράμματος ή/και παράλογα αποτελέσματα

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



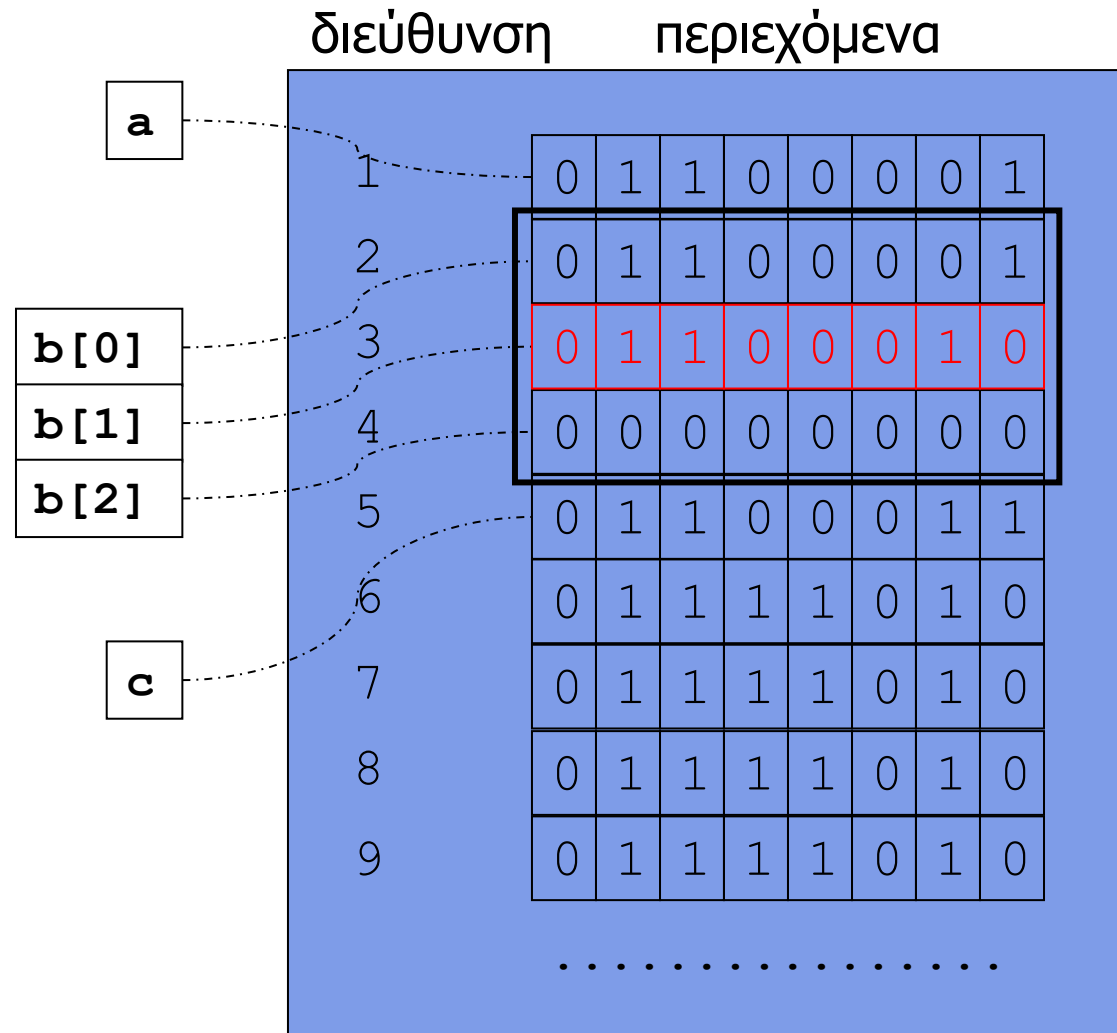

```
char a;  
char b[3];  
char c;  
  
b[0]='a';
```



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

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

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

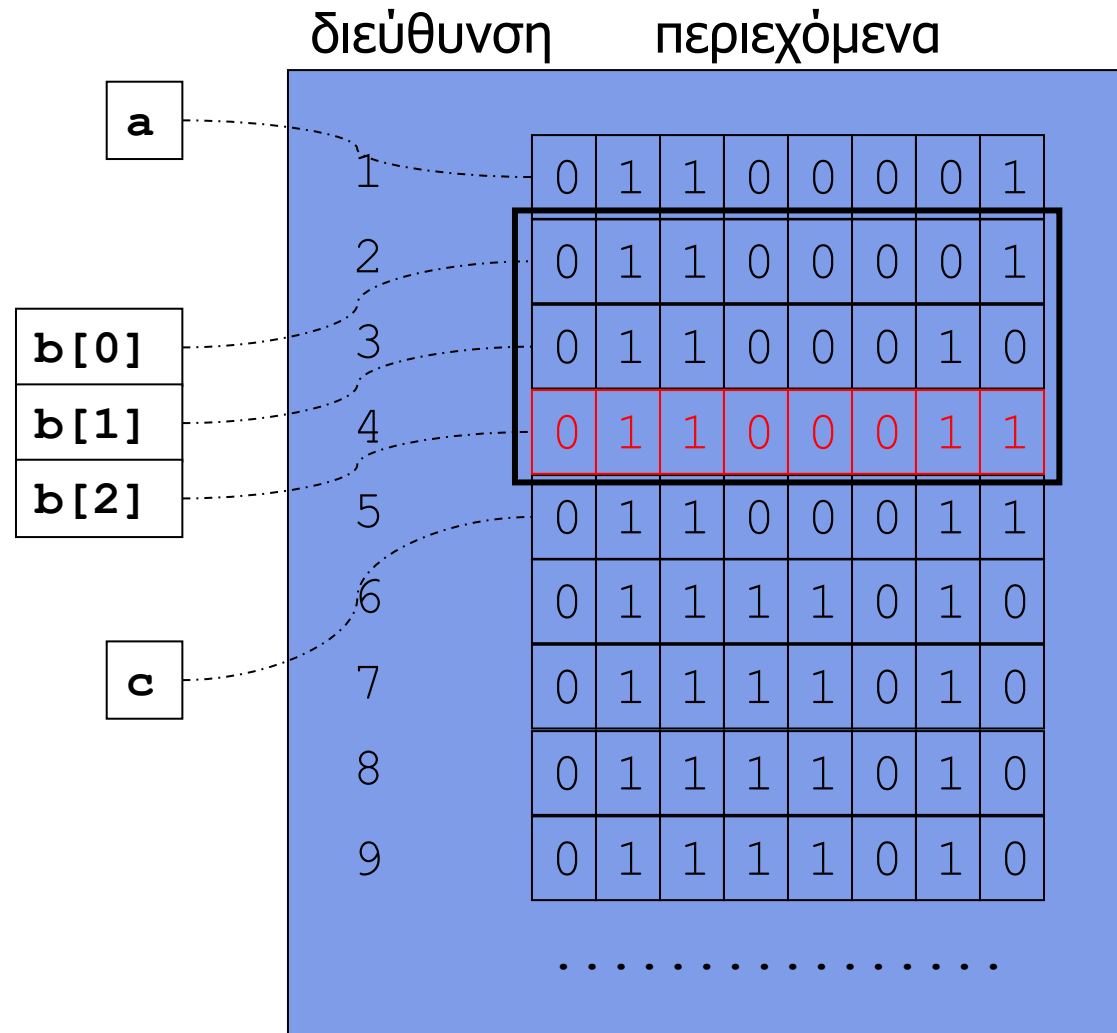


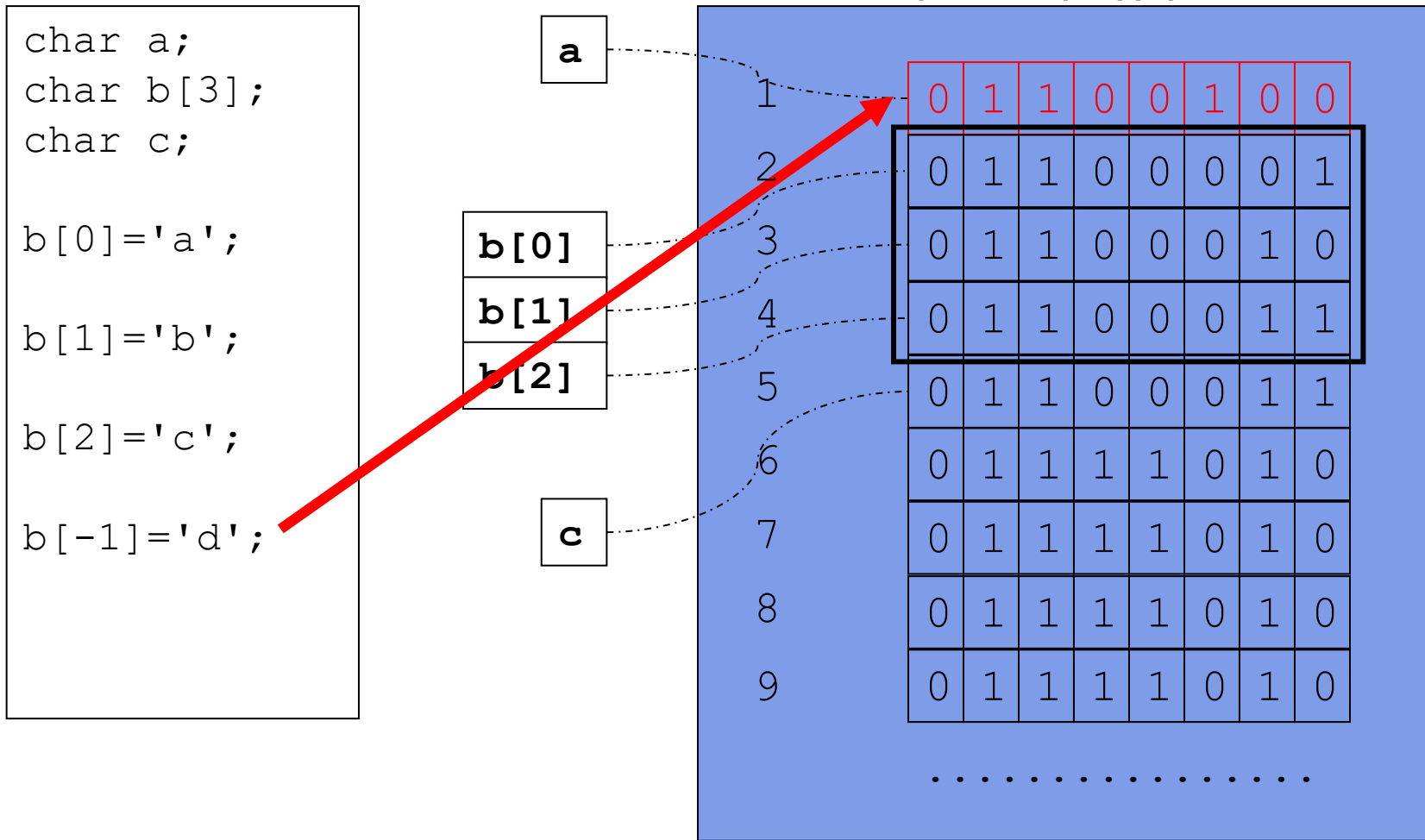
```
char a;  
char b[3];  
char c;
```

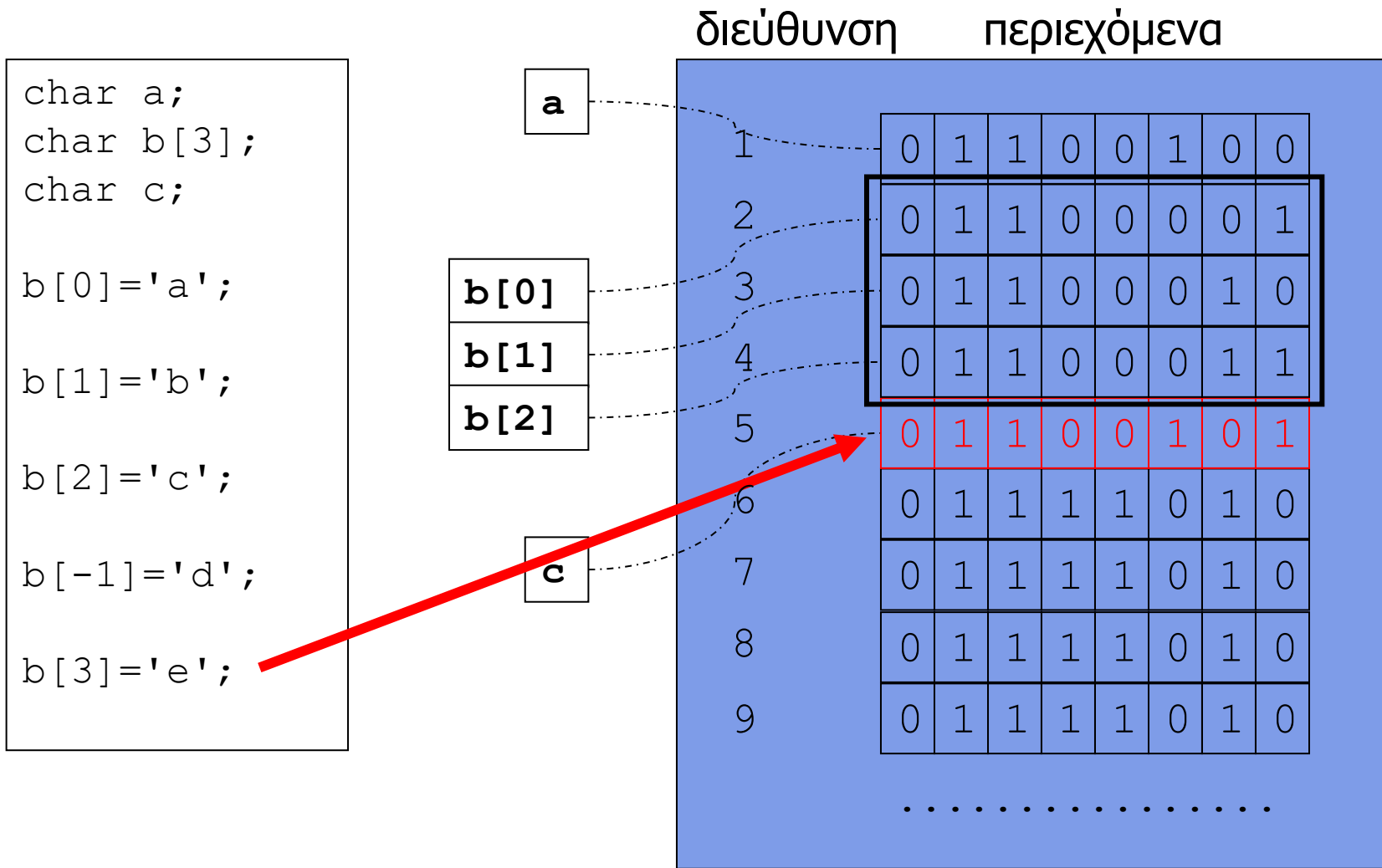
```
b[0]='a';
```

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

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







Τρόποι αυτοματοποιημένου ελέγχου για προσπελάσεις εκτός ορίων πίνακα

- Valgrind

`valgrind --tool=exp-sgcheck <εκτελέσιμο>`

- Σύνολο από εργαλεία που μπορούν να πραγματοποιήσουν ελέγχους (κατά το χρόνο εκτέλεσης) για συχνά σφάλματα
- Ιδιαίτερα καλό σε σφάλματα που αφορούν τη διαχείριση μνήμης (και τις προσπελάσεις εκτός ορίων πίνακα)

- gcc options

`-fsanitize=address`

- Προσθέτει ελέγχους (που εκτελούνται κατά το χρόνο εκτέλεσης) για σφάλματα σχετικά με τη διαχείριση μνήμης

Παραμετροποιημένη προσπάθεια πίνακα

- Ο προσδιορισμός της θέσης των στοιχείου ενός πίνακα μπορεί να γίνει μέσω **οποιασδήποτε** έκφρασης αποτιμάται σε ακέραια τιμή
 - π.χ. την τιμή μιας μεταβλητής
 - πρέπει να κυμαίνεται **μέσα στα όρια** του πίνακα
- Η διάσχιση ενός πίνακα μεγέθους N (για ανάγνωση ή γράψιμο των επιμέρους στοιχείων του) μπορεί να γίνει εύκολα χρησιμοποιώντας μια δομή 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 $[i]$

```
/* ανάγνωση 10 ακεραίων και εκτύπωση αθροίσματος */

#include <stdio.h>
#define N 10

int main(int argc, char* argv[]) {
    int elem[N]; /* πίνακας N ακεραίων */
    int sum;      /* άθροισμα στοιχείων */
    int i;        /* προσπέλαση στοιχείων πίνακα */

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

    sum = 0;
    for (i=0; i<N; i++) {
        sum = sum + elem[i];
    }

    printf("sum is %d\n", sum);
    return(0);
}
```



```

/* ανάγνωση 10 ακεραίων και εκτύπωση μέγιστης τιμής */

#include <stdio.h>
#define N 10

int main(int argc, char* argv[]) {
    int elem[N]; /* πίνακας N ακεραίων */
    int max_i;    /* θέση μεγαλύτερου στοιχείου */
    int i;        /* προσπέλαση στοιχείων πίνακα */

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

    max_i = 0;
    for (i=1; i<N; i++) {
        if (elem[i] > elem[max_i]) {
            max_i = i;
        }
    }

    printf("maximum is %d\n", elem[max_i]);
    return(0);
}

```

```

/* ανάποδη εκτύπωση εισόδου - ως 80 χαρακτήρες ή '\n' */

#include <stdio.h>
#define SIZE 80

int main(int argc, char *argv[]) {
    char buf[SIZE]; /* αποθήκευση χαρακτήρων εισόδου */
    int i;          /* προσπέλαση στοιχείων πίνακα */

    for (i=0; i < SIZE; i++) {
        buf[i] = getchar();
        if (buf[i] == '\n') {
            break;
        }
    }

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

    return(0);
}

```

Πίνακες πολλών διαστάσεων

- Ένας πίνακας μπορεί να δηλωθεί ως N-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.

$$a[N][M] \Leftrightarrow a[N*M]$$

$$a[i][j] \Leftrightarrow a[i*M + j]$$

```
...  
char a[2][3];
```

a[0][0]
<u>a</u>[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]

διεύθυνση περιεχόμενα

1	0	0	1	1	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	0	0	0	0	0	0	0
8	0	1	1	1	1	0	1	0
9	0	1	1	1	1	0	1	0
.								

```

...
char a[2][3];

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

```

a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]

διεύθυνση περιεχόμενα

1	0	0	1	1	0	0	0	1
2	0	1	1	0	0	0	0	1
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	0	0	0	0	0	0	0
8	0	1	1	1	1	0	1	0
9	0	1	1	1	1	0	1	0
.								

```

...
char a[2][3];

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

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

```

a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]

διεύθυνση περιεχόμενα

1	0	0	1	1	0	0	0	1
2	0	1	1	0	0	0	0	1
3	0	1	1	0	0	0	1	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	0	0	0	0	0	0	0
8	0	1	1	1	1	0	1	0
9	0	1	1	1	1	0	1	0
.....								

```

...
char a[2][3];

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

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

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

```

a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]

διεύθυνση περιεχόμενα

1	0	0	1	1	0	0	0	1
2	0	1	1	0	0	0	0	1
3	0	1	1	0	0	0	1	0
4	0	1	1	0	0	0	1	1
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	1	1	1	1	0	1	0
9	0	1	1	1	1	0	1	0
.....								

```

...
char a[2][3];

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

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

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

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

```

a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]

διεύθυνση περιεχόμενα

1	0	0	1	1	0	0	0	1
2	0	1	1	0	0	0	0	1
3	0	1	1	0	0	0	1	0
4	0	1	1	0	0	0	1	1
5	0	1	1	0	0	1	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	1	1	1	1	0	1	0
9	0	1	1	1	1	0	1	0
.								


```

...
char a[2][3];

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

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

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

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

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

```

a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]

διεύθυνση περιεχόμενα

1	0	0	1	1	0	0	0	1
2	0	1	1	0	0	0	0	1
3	0	1	1	0	0	0	1	0
4	0	1	1	0	0	0	1	1
5	0	1	1	0	0	1	0	0
6	0	1	1	0	0	1	0	1
7	0	0	0	0	0	0	0	0
8	0	1	1	1	1	0	1	0
9	0	1	1	1	1	0	1	0
.....								

```

...
char a[2][3];

a[0][0]='a';
a[0][1]='b';
a[0][2]='c';

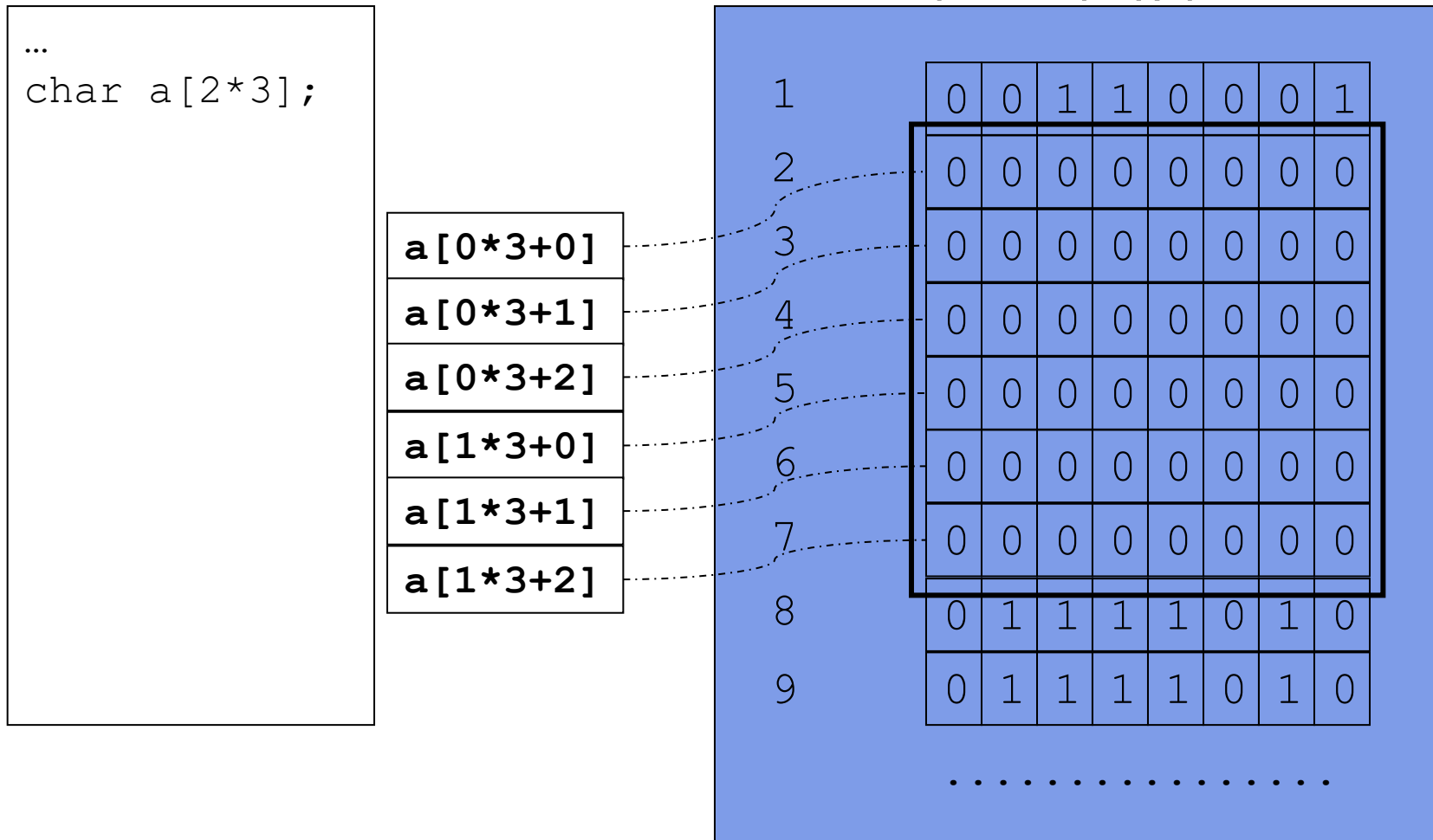
a[1][0]='d';
a[1][1]='e';
a[1][2]='f';

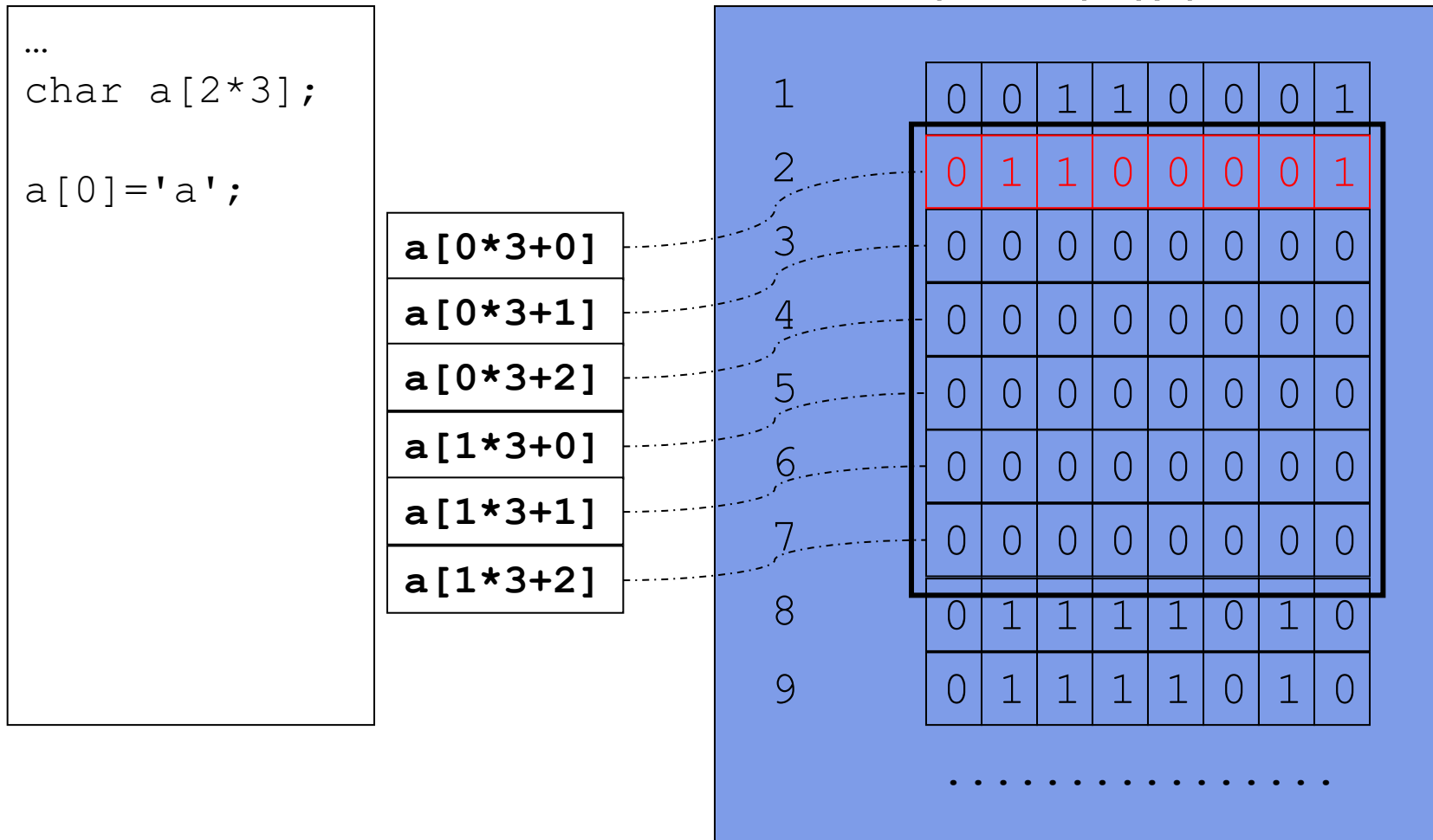
```

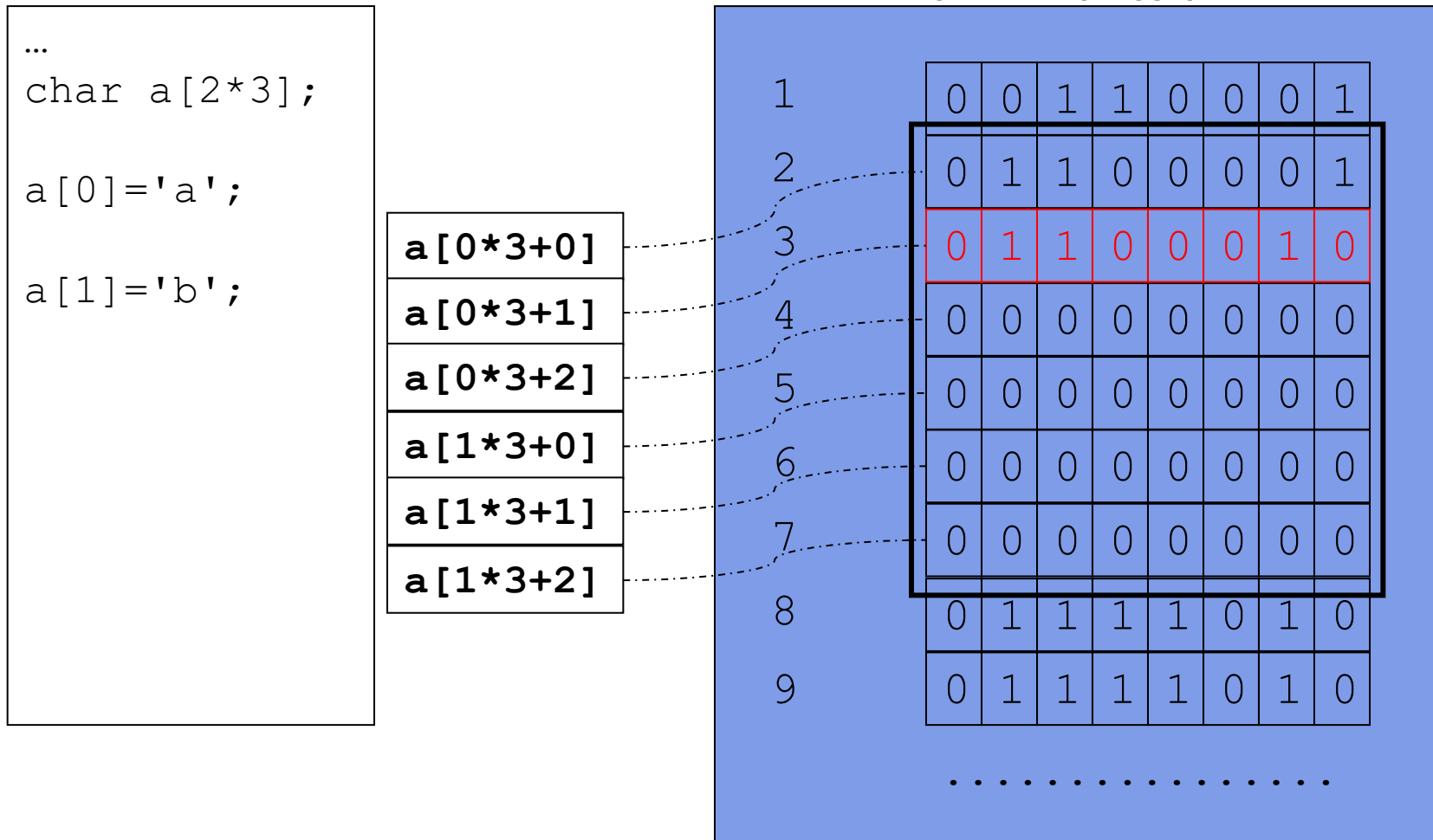
a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]

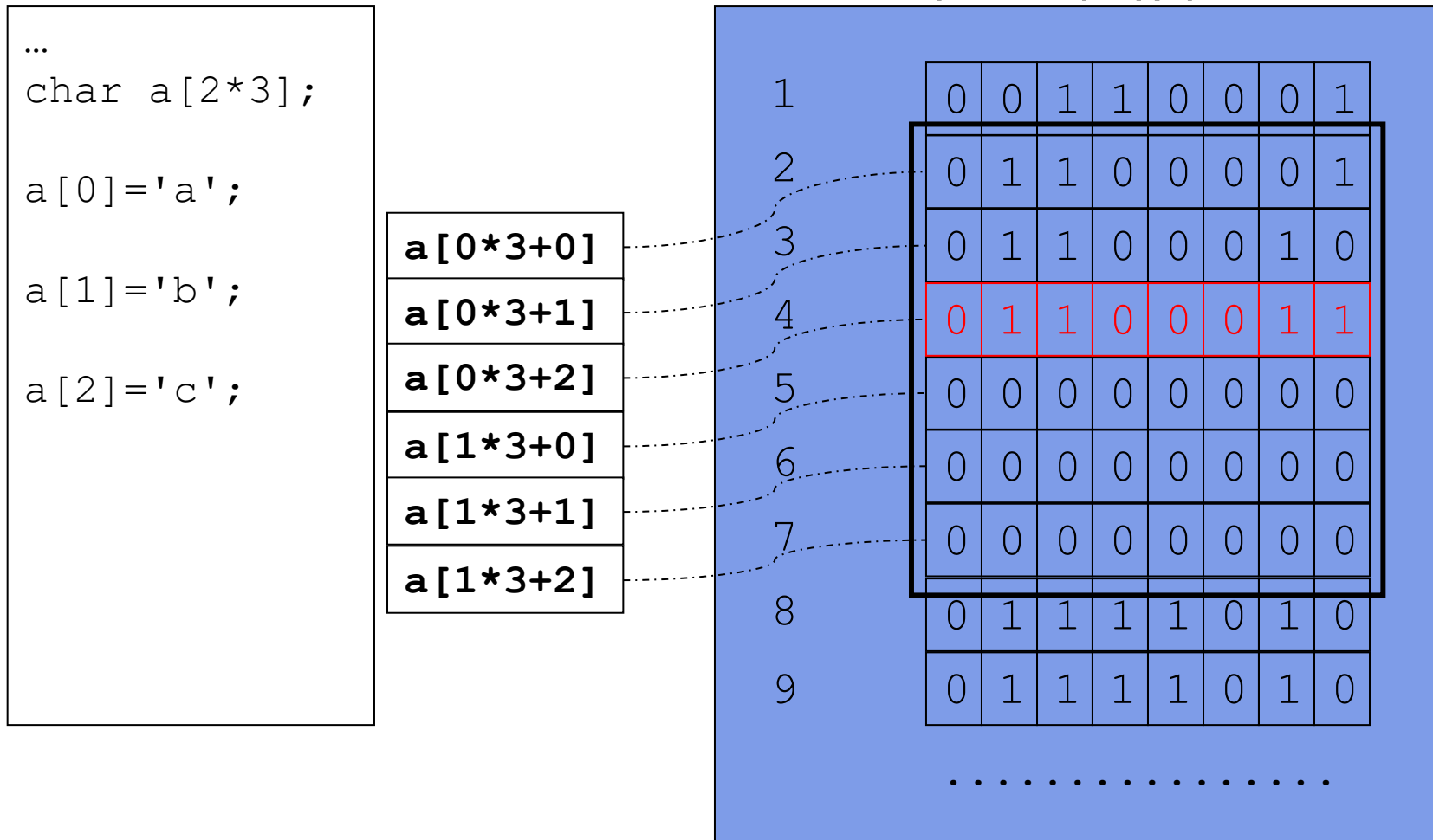
διεύθυνση περιεχόμενα

1	0	0	1	1	0	0	0	1
2	0	1	1	0	0	0	0	1
3	0	1	1	0	0	0	1	0
4	0	1	1	0	0	0	1	1
5	0	1	1	0	0	1	0	0
6	0	1	1	0	0	1	0	1
7	0	1	1	0	0	1	1	0
8	0	1	1	1	1	0	1	0
9	0	1	1	1	1	0	1	0
.								









```

...
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]

διεύθυνση περιεχόμενα

1	0	0	1	1	0	0	0	1
2	0	1	1	0	0	0	0	1
3	0	1	1	0	0	0	1	0
4	0	1	1	0	0	0	1	1
5	0	1	1	0	0	1	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	1	1	1	1	0	1	0
9	0	1	1	1	1	0	1	0
.....								

```

...
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]

διεύθυνση περιεχόμενα

1	0	0	1	1	0	0	0	1
2	0	1	1	0	0	0	0	1
3	0	1	1	0	0	0	1	0
4	0	1	1	0	0	0	1	1
5	0	1	1	0	0	1	0	0
6	0	1	1	0	0	1	0	1
7	0	0	0	0	0	0	0	0
8	0	1	1	1	1	0	1	0
9	0	1	1	1	1	0	1	0
.								


```

...
char a[2*3];

a[0]='a';

a[1]='b';

a[2]='c';

a[3]='d';

a[4]='e';

a[5]='f';

```

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

διεύθυνση περιεχόμενα

1	0	0	1	1	0	0	0	1
2	0	1	1	0	0	0	0	1
3	0	1	1	0	0	0	1	0
4	0	1	1	0	0	0	1	1
5	0	1	1	0	0	1	0	0
6	0	1	1	0	0	1	0	1
7	0	1	1	0	0	1	1	0
8	0	1	1	1	1	0	1	0
9	0	1	1	1	1	0	1	0
.								

Αρχικοποίηση πινάκων

- Υποστηρίζεται αρχικοποίηση κατά την δήλωση
 - `int a[4]={1,2,3,4};`
- Αν η διάσταση αφαιρεθεί απροσδιόριστη, ορίζεται **εμμέσως** από τον αριθμό των αρχικών τιμών
 - `int a[]={1,2,3,4};`
- Αν η διάσταση προσδιοριστεί ρητά αλλά δοθούν λιγότερες αρχικές τιμές, οι **υπόλοιπες** είναι 0
 - `int a[4]={1,2}; /* a[2],a[3] είναι 0 */`
- Αν αντί για τιμές μεμονωμένων στοιχείων δοθεί απλά η (ειδική) τιμή 0 ή `'\0'`, **όλα** τα στοιχεία γίνονται 0
 - `int a[4]={0}; /* όλα τα στοιχεία 0 */`
 - `char c[4]=''\0''; /* όλα τα στοιχεία '\0' */`
- Χωρίς καμία αρχικοποίηση, ο πίνακας έχει **τυχαίες** τιμές
 - όπως άλλωστε ισχύει για οποιαδήποτε μεταβλητή ...

Αρχικοποίηση πολυδιάστατων πινάκων

- Οι τιμές κάθε διάστασης δίνονται ανάμεσα σε { }
 - `int c[2][3]={ {1,2,3}, {4,5,6} };`
- Μόνο η **πρώτη** διάσταση μπορεί να προσδιοριστεί **έμμεσα** μέσω της έκφρασης αρχικοποίησης
 - `int a[][3] ={{1,2,3}, {4,5,6}};`
- Για κάθε διάσταση ισχύουν οι συμβατικοί κανόνες αρχικοποίησης (βλέπε προηγούμενη διαφάνεια)
 - `int c[2][3]={ {1}, {4,5,6} };`
 - `int a[2][3]={ {0}, {4,5,6} };`
- Αν μια διάσταση δεν αρχικοποιείται ρητά, τότε **όλα** αντίστοιχα στοιχεία λαμβάνουν **αυτομάτως** την τιμή 0
 - `int a[2][3]={ {1,2,3} }; /* η 2η γραμμή είναι 0 */`
 - `int a[2][3]={ {0} }; /* όλες οι γραμμές 0 */`

Πρόσθεση πινάκων

$$\begin{array}{c} a \\ \begin{bmatrix} 1 & 2 & 3 \\ 0 & -5 & 1 \\ 3 & 1 & 7 \end{bmatrix} \end{array} + \begin{array}{c} b \\ \begin{bmatrix} 1 & 3 & -4 \\ 2 & 5 & 0 \\ 4 & 5 & -2 \end{bmatrix} \end{array} = \begin{array}{c} c \\ \begin{bmatrix} 2 & 5 & -1 \\ 2 & 0 & 1 \\ 7 & 6 & 5 \end{bmatrix} \end{array}$$

- Αθροίζουμε στοιχείο προς στοιχείο

$$c[i][j] = a[i][j] + b[i][j]$$

- Τα i και j κυμαίνονται από την αρχή έως το τέλος της κάθετης και οριζόντιας διάστασης των πινάκων

```

/* πρόσθεση πινάκων */

#include <stdio.h>
#define N 3

int main(int argc, char *argv[]) {
    int a[N][N] = {{1,2,3},{0,-5,1},{3,1,7}};
    int b[N][N] = {{1,3,-4},{2,5,0},{4,5,-2}};
    int c[N][N]; /* αποθήκευση αποτελέσματος */
    int i,j;      /* προσπέλαση στοιχείων */

    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }

    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            printf("%3d ", c[i][j]);
        }
        printf("\n");
    }

    return(0);
}

```

Πολλαπλασιασμός πινάκων

$$\begin{array}{c} a \\ \begin{bmatrix} 1 & 2 & 3 \\ 0 & -5 & 1 \\ 3 & 1 & 7 \end{bmatrix} \end{array} * \begin{array}{c} b \\ \begin{bmatrix} 1 & 3 & -4 \\ 2 & 5 & 0 \\ 4 & 5 & -2 \end{bmatrix} \end{array} = \begin{array}{c} c \\ \begin{bmatrix} 17 & 28 & -10 \\ -6 & -20 & -2 \\ 33 & 49 & -26 \end{bmatrix} \end{array}$$

- Το εσωτερικό γινόμενο ανάμεσα στην εκάστοτε σειρά $a[i][k]$ και στήλη $b[k][j]$

$$c[i][j] = \sum_k a[i][k] * b[k][j]$$

- Το k κυμαίνεται από την αρχή έως το τέλος της διάστασης πολλαπλασιασμού των πινάκων
 - πλάτος πρώτου πίνακα = ύψος δεύτερου πίνακα

```

/* πολλαπλασιασμός πινάκων */

#include <stdio.h>
#define N 3

int main(int argc, char *argv[]) {
    int a[N][N] = {{1,2,3},{0,-5,1},{3,1,7}};
    int b[N][N] = {{1,3,-4},{2,5,0},{4,5,-2}};
    int c[N][N]; /* αποθήκευση αποτελέσματος */
    int i,j,k;    /* προσπέλαση στοιχείων */

    for (i=0; i<N; i++)
        for (j=0; j<N; j++) {
            c[i][j] = 0;
            for (k=0; k<N; k++) {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }

    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            printf("%3d ", c[i][j]);
        }
        printf("\n");
    }

    return(0);
}

```

strings

Συμβολοσειρές (strings)

- Τα strings υλοποιούνται ως **πίνακες** από `char`
- **Τουλάχιστον** ένα στοιχείο έχει την τιμή `'\0'` (0)
- Το **πρώτο** στοιχείο με αυτή την τιμή καθορίζει το τέλος του string – ονομάζεται **τερματικό**
 - το τερματικό **δεν** είναι απαραίτητα/αναγκαστικά το τελευταίο στοιχείο του πίνακα χαρακτήρων
 - τα στοιχεία μετά το τερματικό **δεν** λαμβάνονται υπόψη στις διάφορες λειτουργίες που αφορούν τα strings
- Ένα πρόγραμμα που χρησιμοποιεί strings μπορεί να **υποθέσει** πως αυτά τερματίζονται με `'\0'`
- Ένα πρόγραμμα που δημιουργεί strings πρέπει να **φροντίσει** αυτά να τερματίζονται με `'\0'`

```
char str1[] = {'w', 'i', 'n'};    /* not a string! */

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

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

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

char str5[7] = {'\0'};

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 & εκτύπωση strings

- Μπορούμε να τα υλοποιήσουμε (εμείς) χαρακτήρα προς χαρακτήρα μέσω `getchar()` και `putchar()`
- Πιο απλά, χρησιμοποιούμε `scanf()` και `printf()`
 - με **προσδιοριστή** `%s` και παράμετρο πίνακα χαρακτήρων
 - στην `scanf` **δεν** βάζουμε `&` πριν το όνομα της μεταβλητής
 - το γιατί θα το δούμε λίγο αργότερα ...
- Η `printf()` σταματά την εκτύπωση των περιεχομένων του πίνακα όταν βρει το τερματικό
- Η `scanf()` σταματά την ανάγνωση χαρακτήρων (και την αποθήκευση τους στον πίνακα) **μόνο** όταν βρεθεί ένας «λευκός» χαρακτήρας
 - **προσοχή: μπορεί να γίνει αποθήκευση εκτός ορίων του πίνακα που δίνεται ως παράμετρος!**

προσδιοριστής string

```
#include<stdio.h>

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

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

    return(0);
}
```

δίνεται το όνομα της μεταβλητής
(του πίνακα) **χωρίς** το &

αν ο χρήστης εισάγει συμβολοσειρά με περισσότερους από
3 χαρακτήρες, αυτοί (ή/και το τερματικό) θα αποθηκευτούν
σε θέσεις μνήμης που βρίσκονται **εκτός ορίων** του πίνακα

...

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

```
#include<stdio.h>

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

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

    return(0);
}
```

```
/* μήκος string str */

#include <stdio.h>
#define N 32

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

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

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

    printf("length of %s is %d\n", str, len);

    return(0);
}
```

```

/* ανάποδη αντιγραφή του str1 στο str2 */

#include <stdio.h>
#define N 32

int main(int argc, char *argv[]) {
    int len, pos;
    char str1[N], str2[N];

    printf("enter string: ");
    scanf("%31s", str1);

    /* βρες το τέλος του str1 */
    for (len=0; str1[len]!='\0'; len++) {}

    /* αντιέγραψε ανάποδα στο str2 */
    for (pos=0, len--; len>=0; pos++, len--) {
        str2[pos] = str1[len];
    }
    str2[pos] = '\0'; /* βάλε τερματικό */

    printf("%s in reverse is %s\n", str1, str2);

    return(0);
}

```

```

/* αντιγραφή του str1 "και" του str2 στο str3 */
#include <stdio.h>
#define N 32

int main(int argc, char *argv[]) {
    char str1[N], str2[N], str3[2*N-1];
    int i, j;

    printf("enter string: ");
    scanf("%31s", str1);
    printf("enter string: ");
    scanf("%31s", str2);

    /* αντιέγραψε το str1 στο str3 */
    for (i=0; str1[i]!='\0'; i++)
        str3[i] = str1[i];

    /* αντιέγραψε το str2 στο str3 */
    for (j=0; str2[j]!='\0'; j++,i++)
        str3[i] = str2[j];

    str3[i]='\0'; /* βάλε τερματικό */

    printf("%s + %s = %s\n", str1, str2, str3);
    return(0);
}

```