

Προγραμματισμός I (HY120)

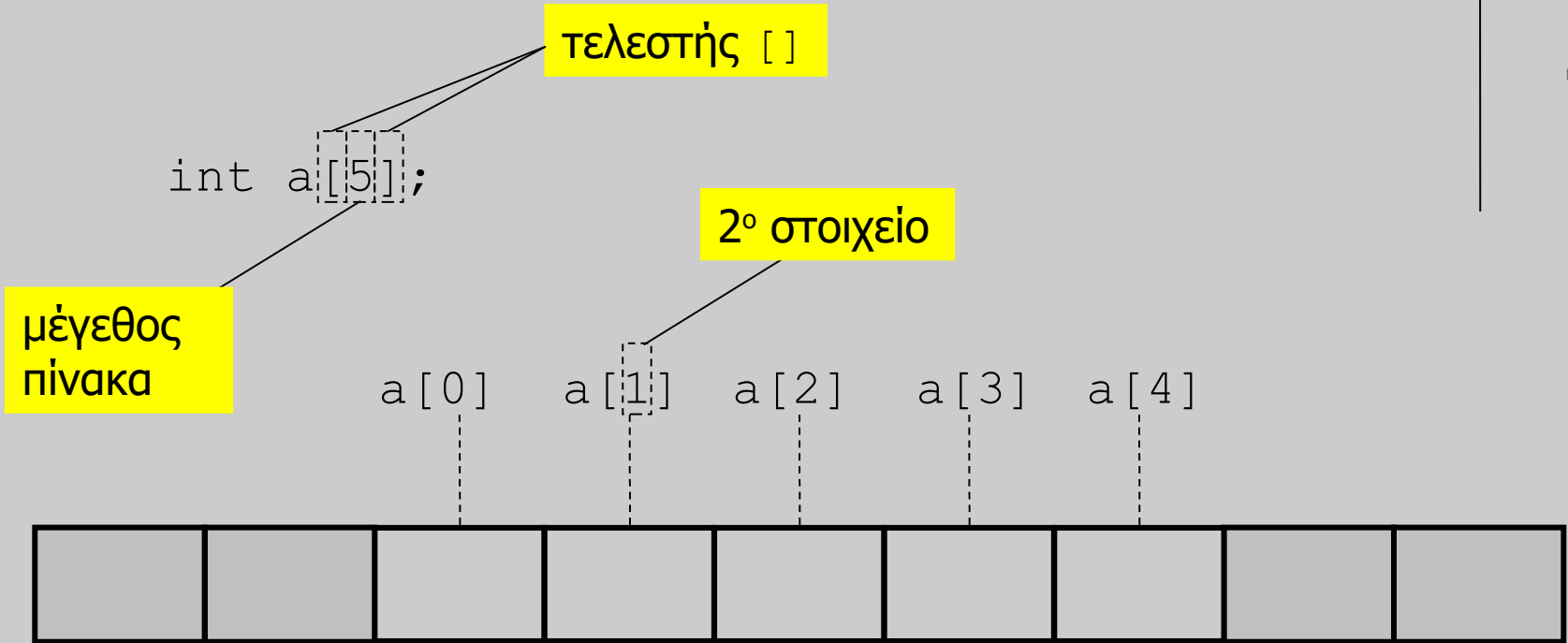
Διάλεξη 8:
Πίνακες, Αλφαριθμητικά



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 αποθηκευτούν **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.



Πρόσβαση σε στοιχεία του πίνακα



4

- 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.
 - Ο μεταφραστής **δεν ελέγχει** αν η θέση που δίνει ο προγραμματιστής είναι εντός των ορίων.



```
int a;  
int b[3];  
int c[] = {5,8,2};
```

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

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

```
b[0] = c[1];
```

```
c[0]++;
```

```
b[--a] = 3;
```

```
a = c[-1];
```

```
b[3] = 5;
```

```
/* ακέραιος */
```

```
/* πίνακας 3 ακεραίων */
```

```
/* πίνακας 3 ακεραίων,  
με τιμές 5, 8, 2 */
```

```
/* a γίνεται ? */
```

```
/* a γίνεται 2 */
```

```
/* b[0] γίνεται 8 */
```

```
/* c[0] γίνεται 6 */
```

```
/* a γίνεται 1, b[1] γίνεται 3 */
```

```
/* !? */
```

```
/* !? */
```



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.
- Μνήμη για την αποθήκευση των στοιχείων ενός πίνακα μεγέθους N από δεδομένα τύπου T :
 - $N * \text{sizeof}(T)$
 - Δεσμεύεται **μονομιάς** ...
 - ... ανεξάρτητα με το ποια στοιχεία του πίνακα θα χρησιμοποιηθούν τελικά από το πρόγραμμα).

Ασφάλεια προσπέλασης μνήμης

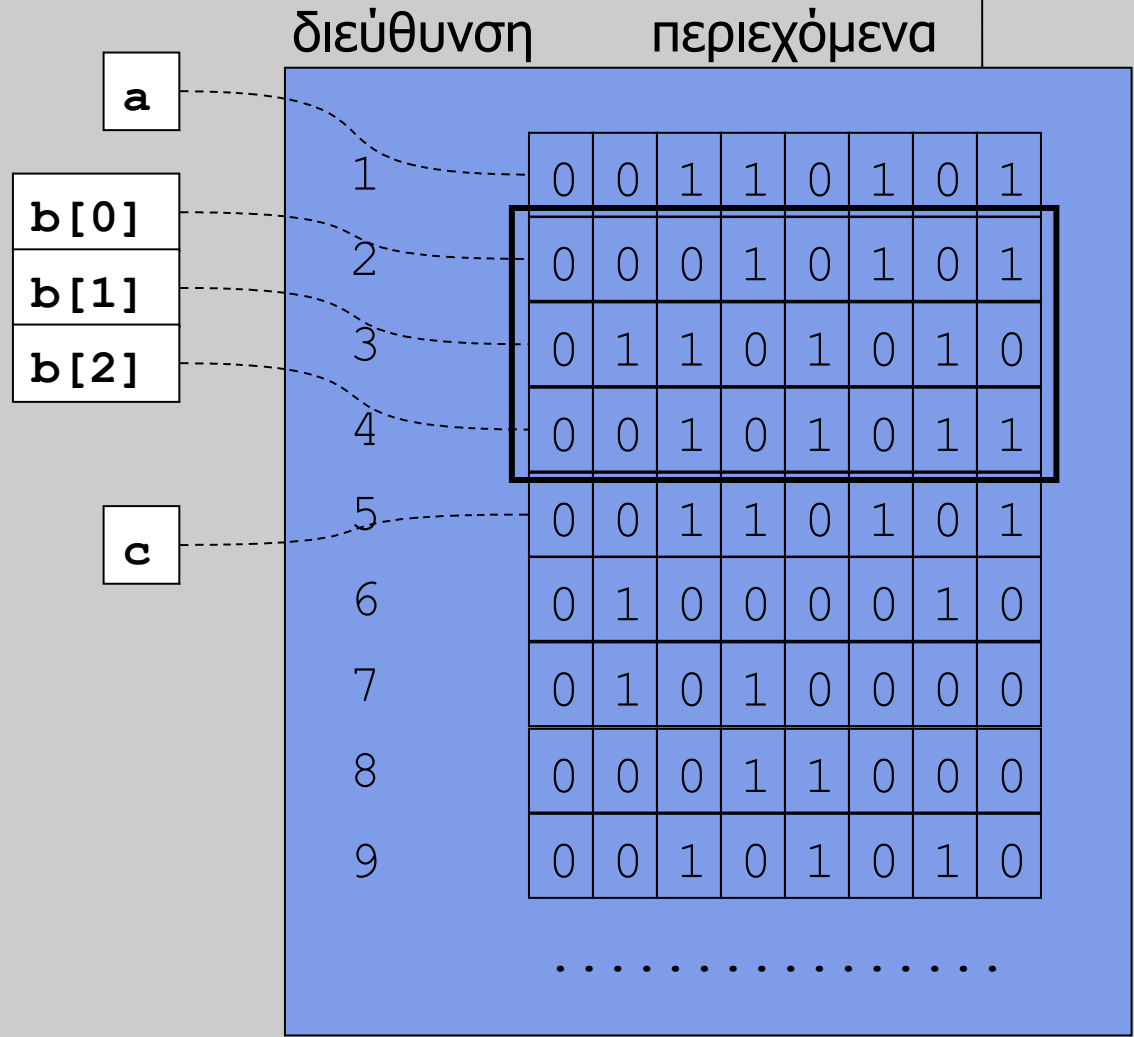


7

- 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.
 - Υπάρχει περίπτωση το πρόγραμμα να «καταστρέφει» δεδομένα **δικών του** μεταβλητών.



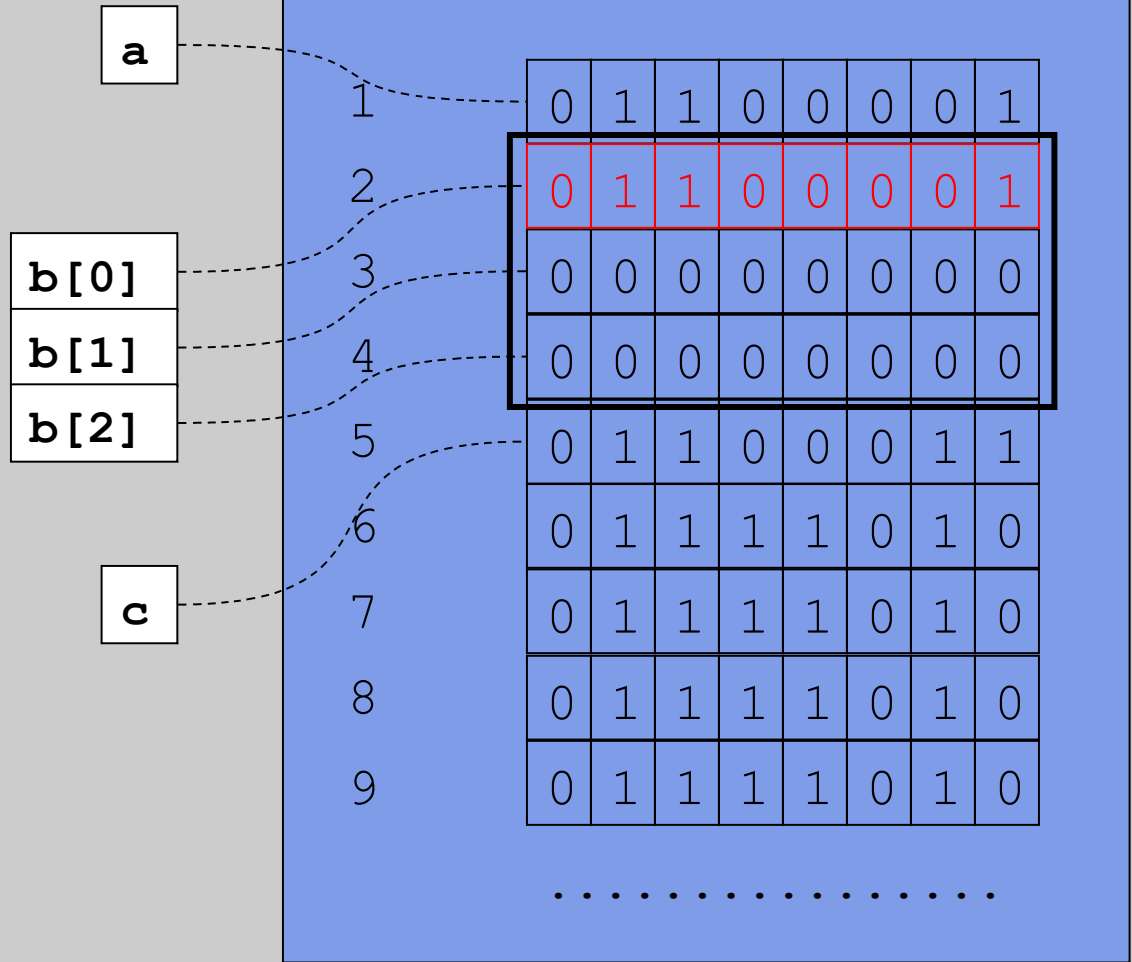
```
char a;  
char b[3];  
char c;
```





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

διεύθυνση περιεχόμενα

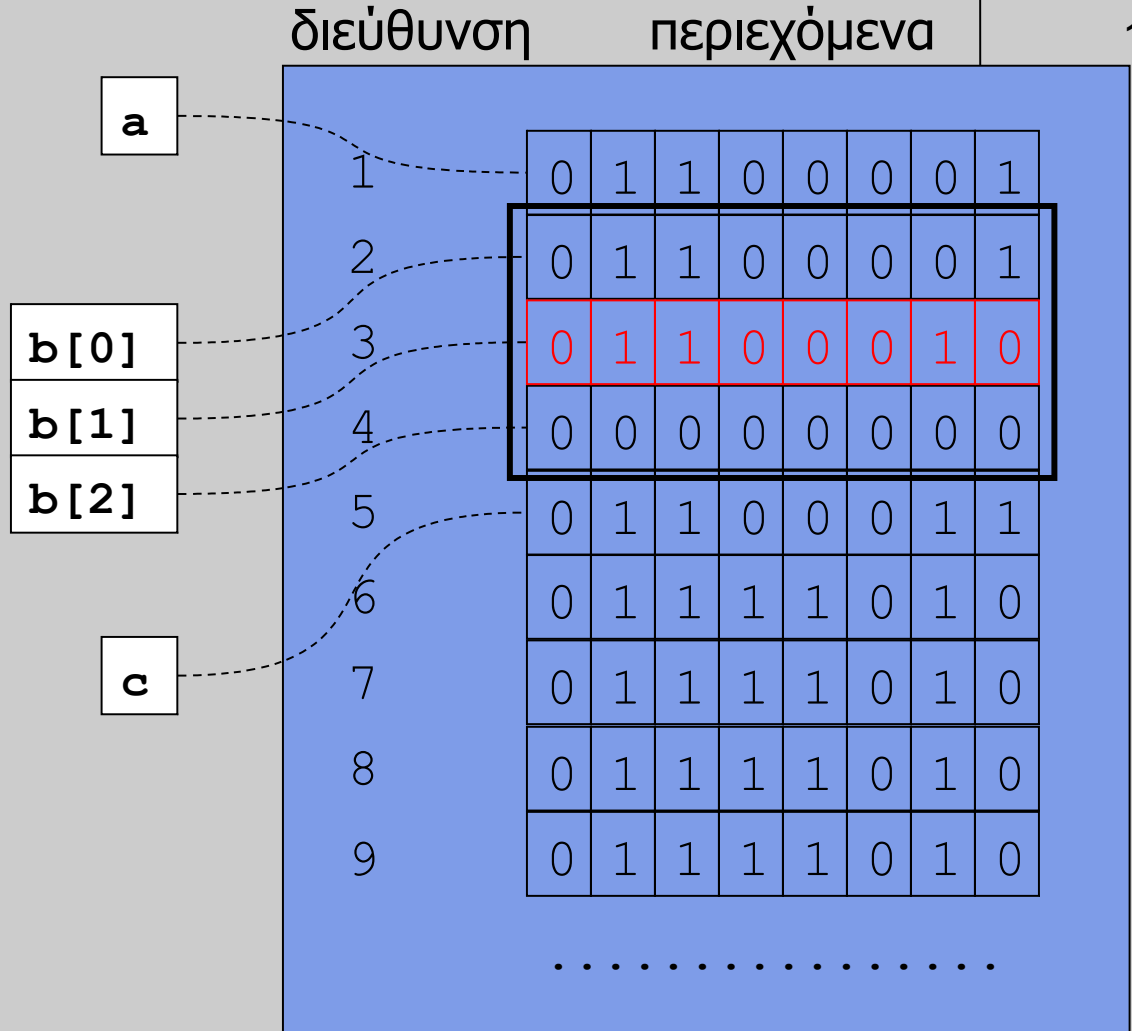




```
char a;  
char b[3];  
char c;
```

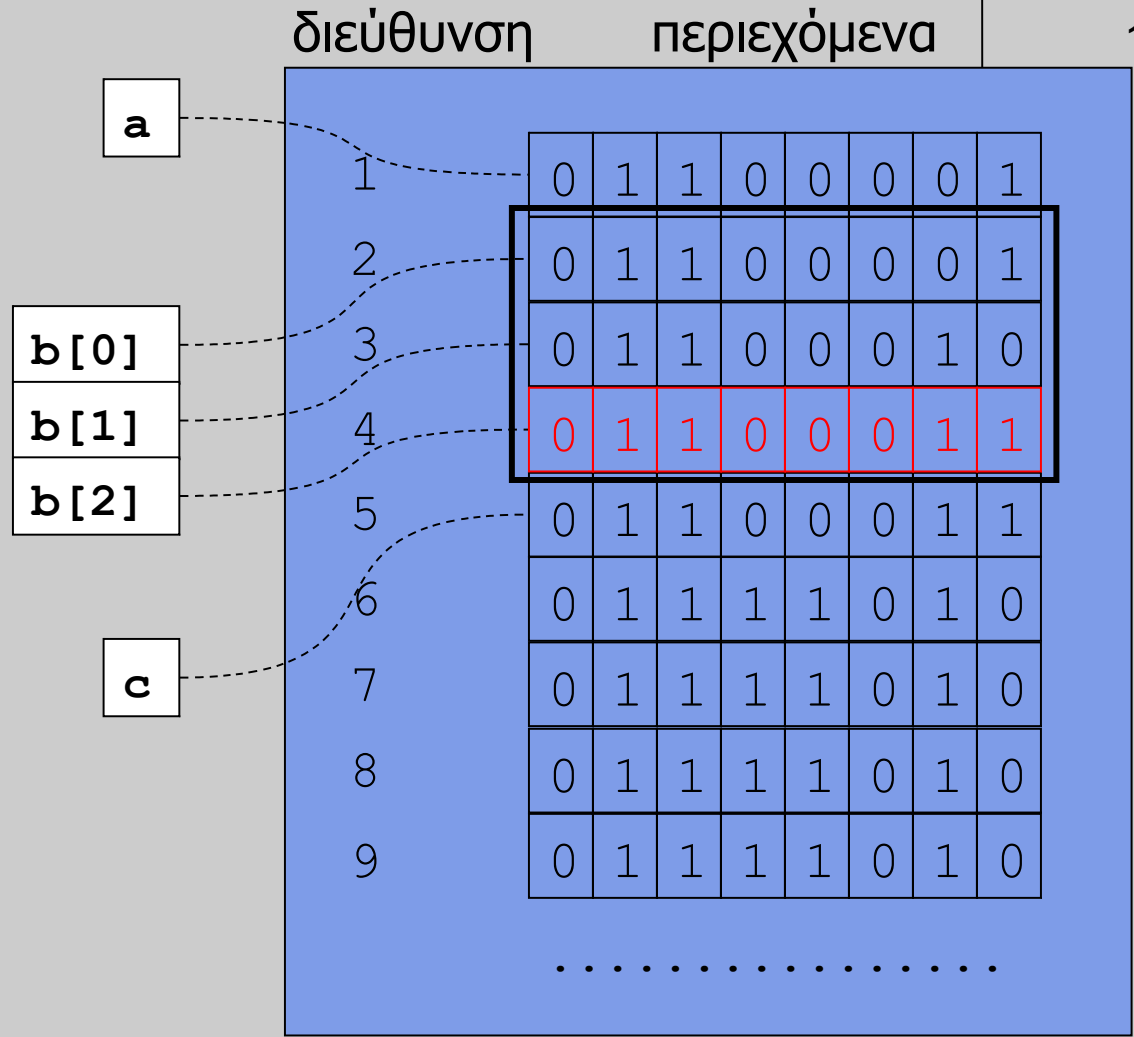
```
b[0]='a';
```

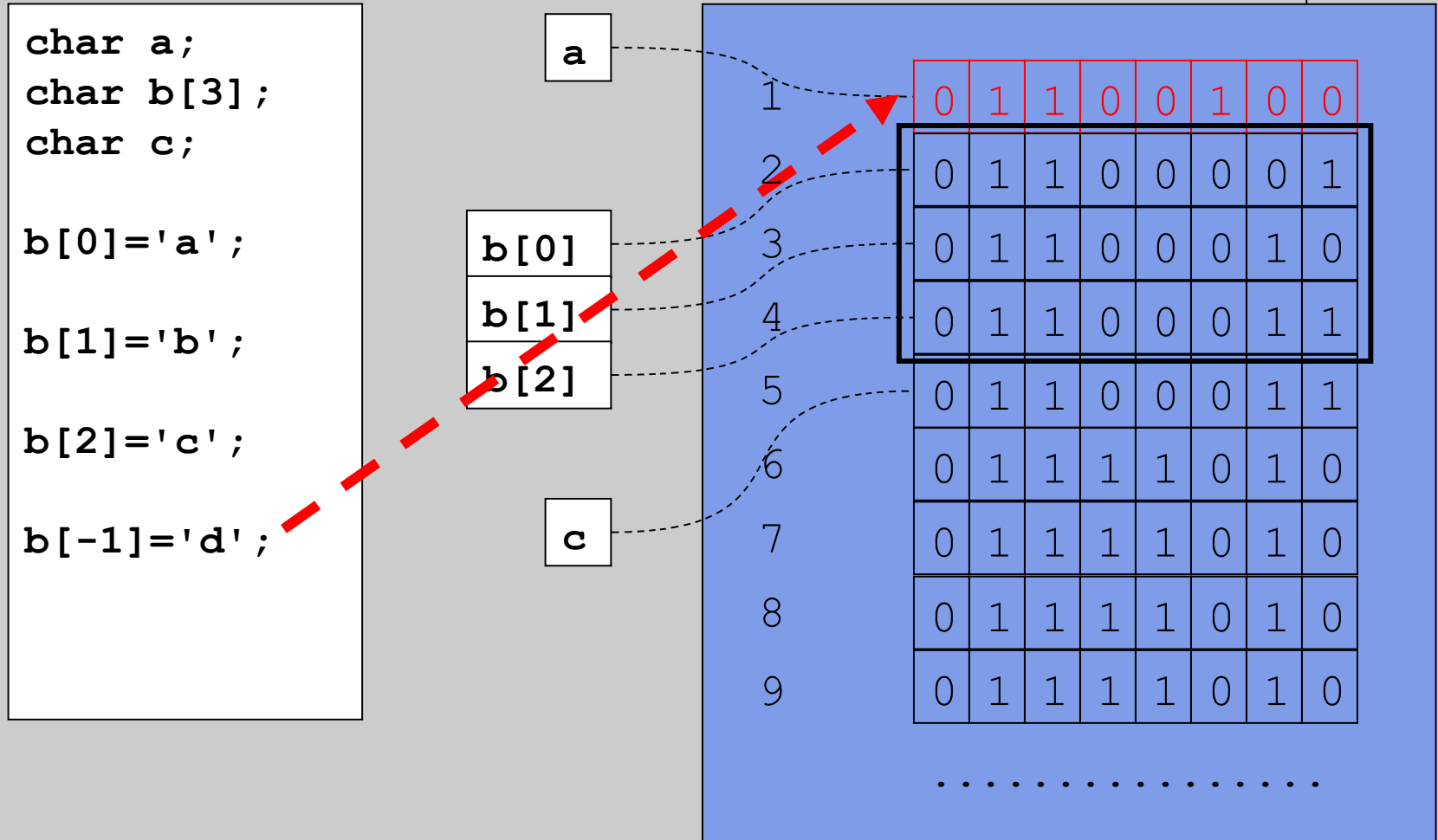
```
b[1]='b';
```

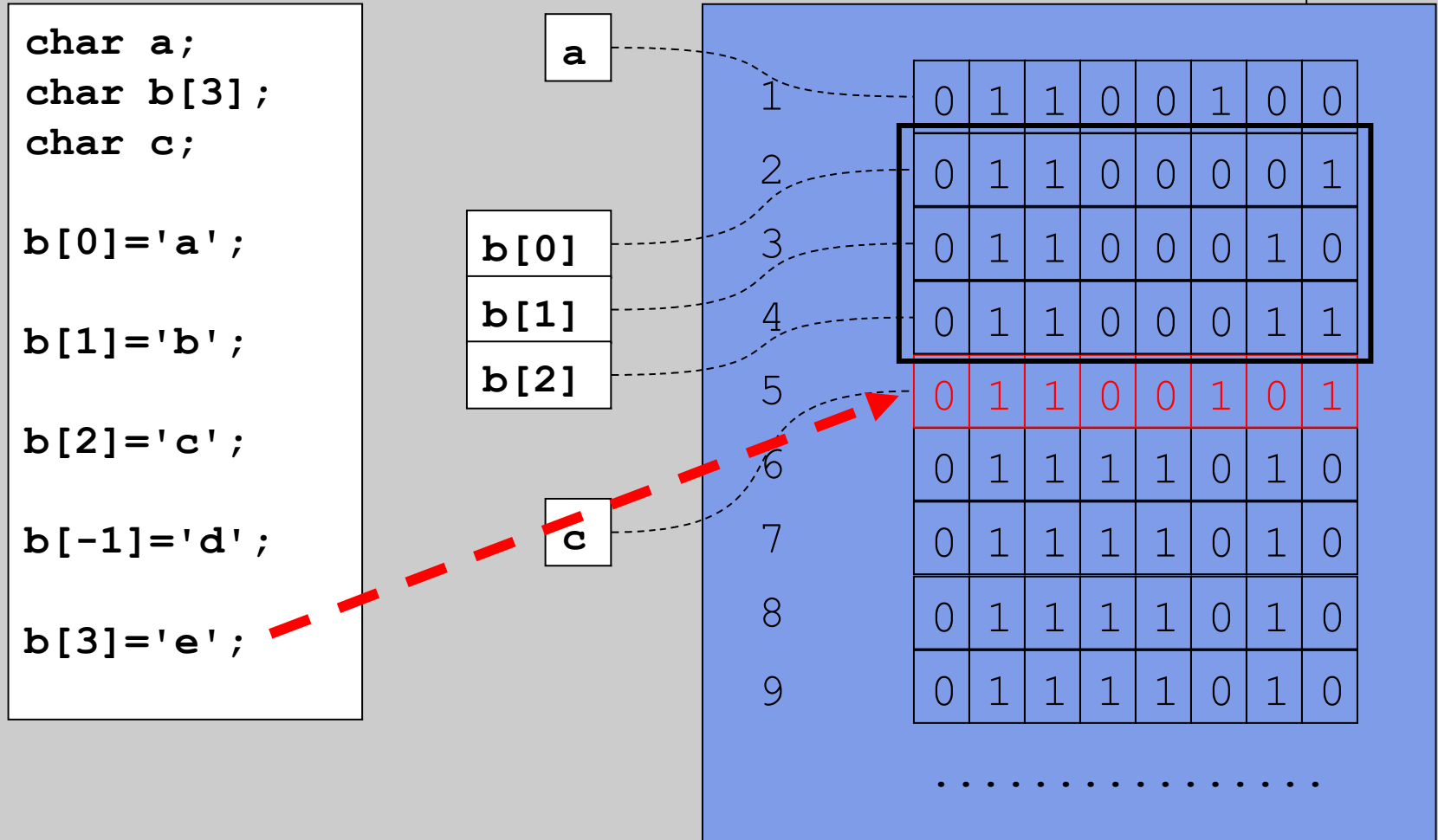




```
char a;  
char b[3];  
char c;  
  
b[0]='a';  
b[1]='b';  
b[2]='c';
```









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.
 - Π.χ. 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 $a[i]$



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

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

    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 /* όπου "N" αντικατέστησε με "10" */

int main(int argc, char* argv[]) {
    int elem[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", &elem[i]);
    }

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

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

```





```
/* ανάποδη εκτύπωση εισόδου - ως 79 χαρακτήρες ή '\n' */
#include <stdio.h>
#define N 80

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

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

    pos--;
    while (pos >= 0) {
        putchar(buf[pos]);
        pos--;
    }
    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.

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

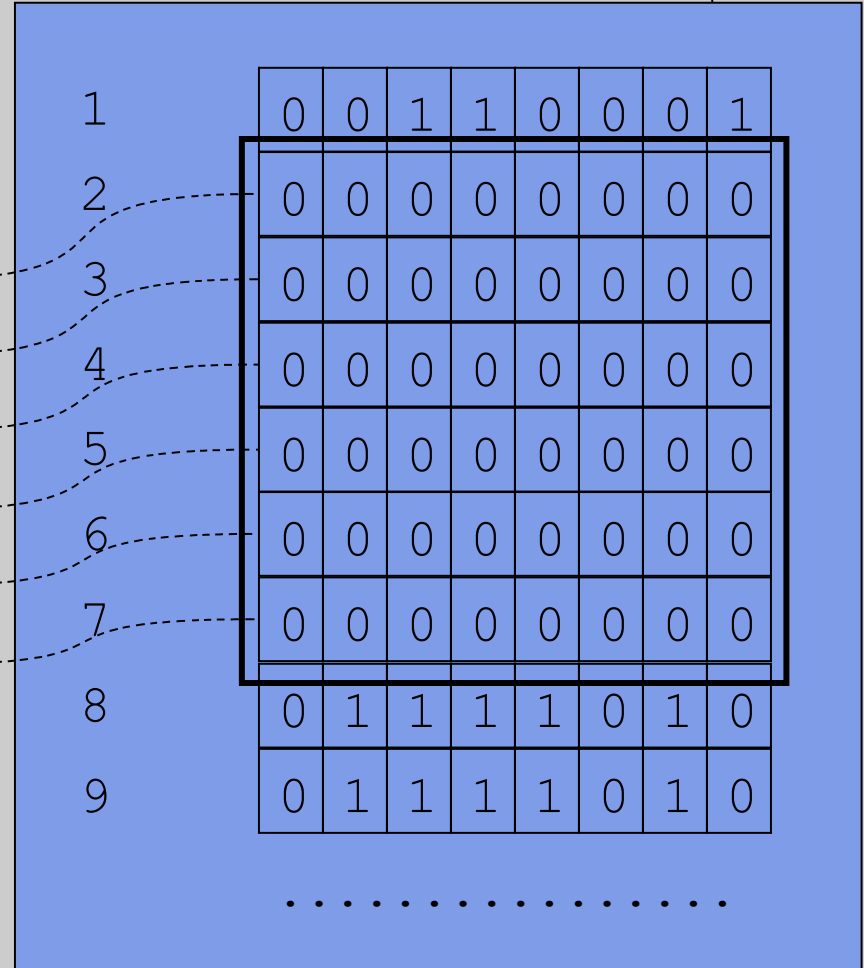
$$a[i][j] \Leftrightarrow b[i*M + j].$$



διεύθυνση περιεχόμενα

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

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

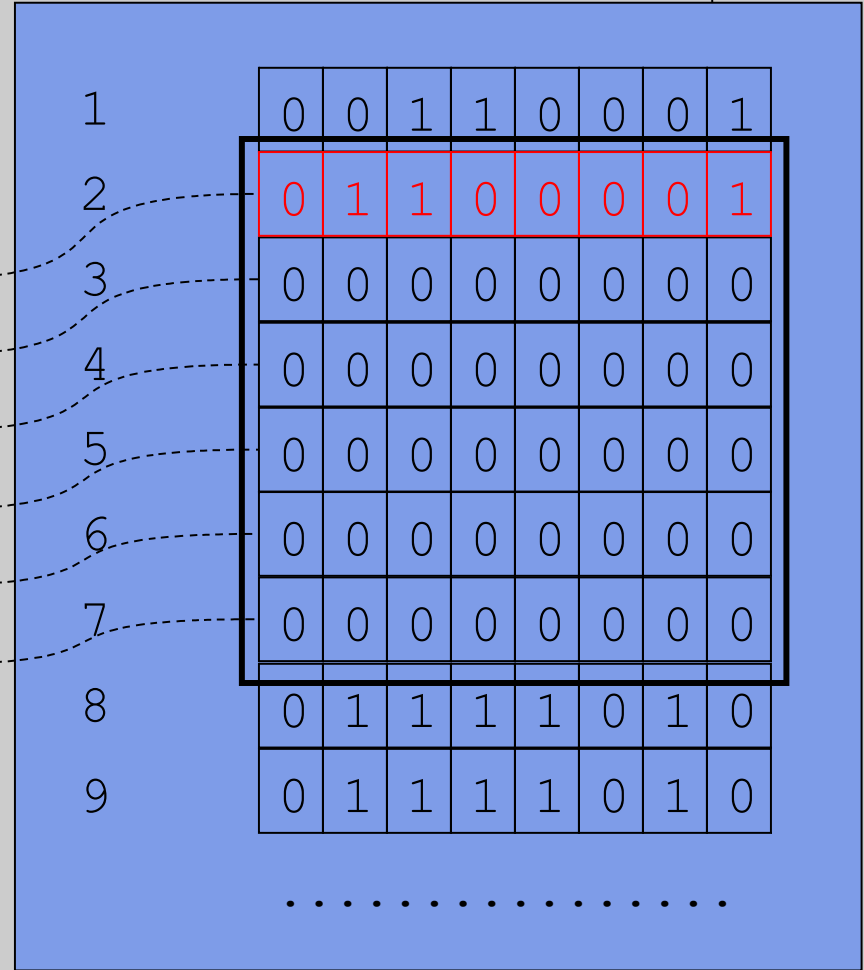




διεύθυνση περιεχόμενα

```
...  
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]

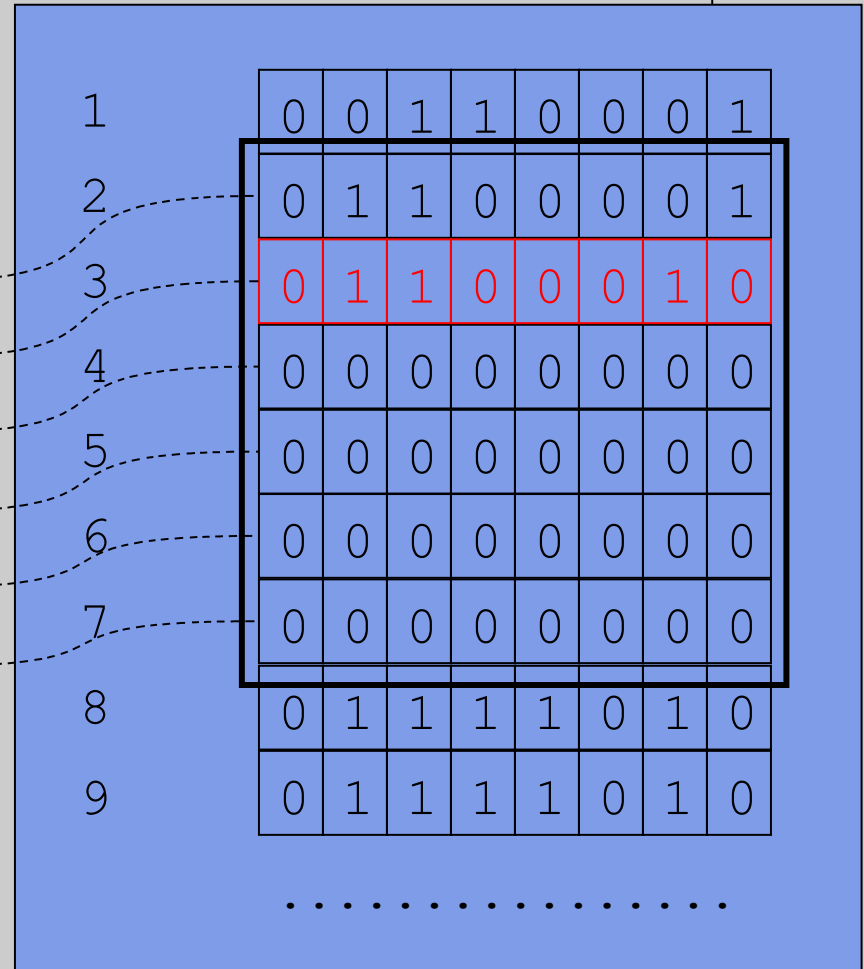




διεύθυνση περιεχόμενα

```
...  
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]

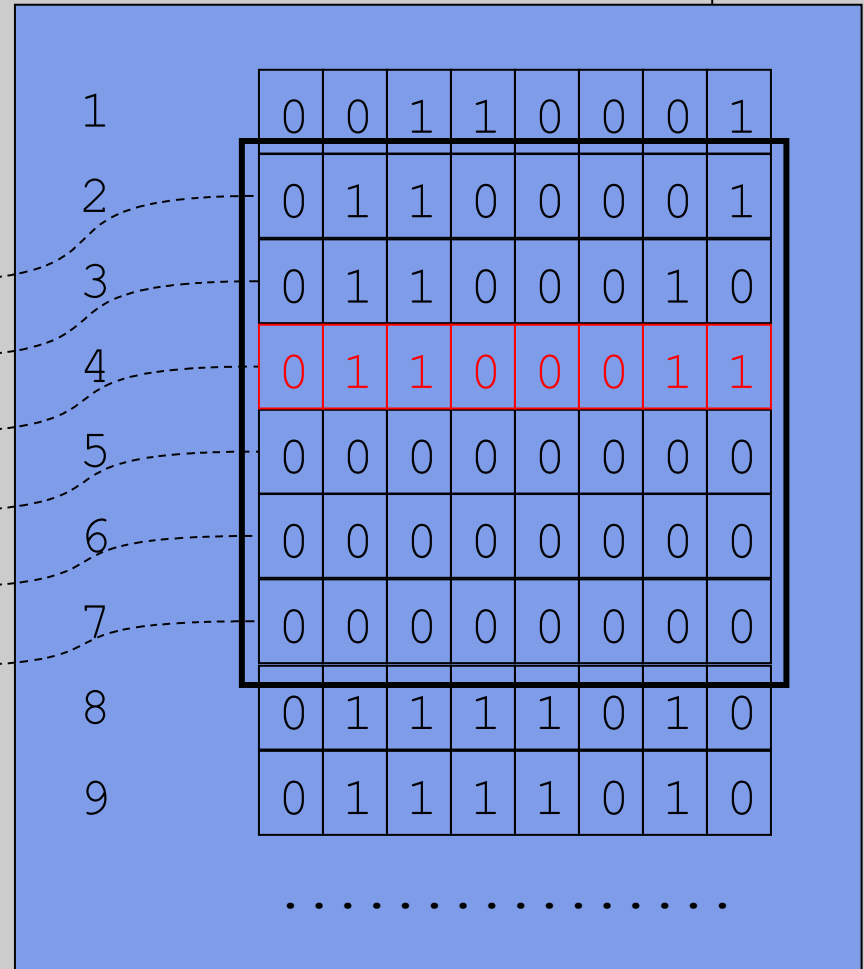




διεύθυνση περιεχόμενα

```
...  
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]

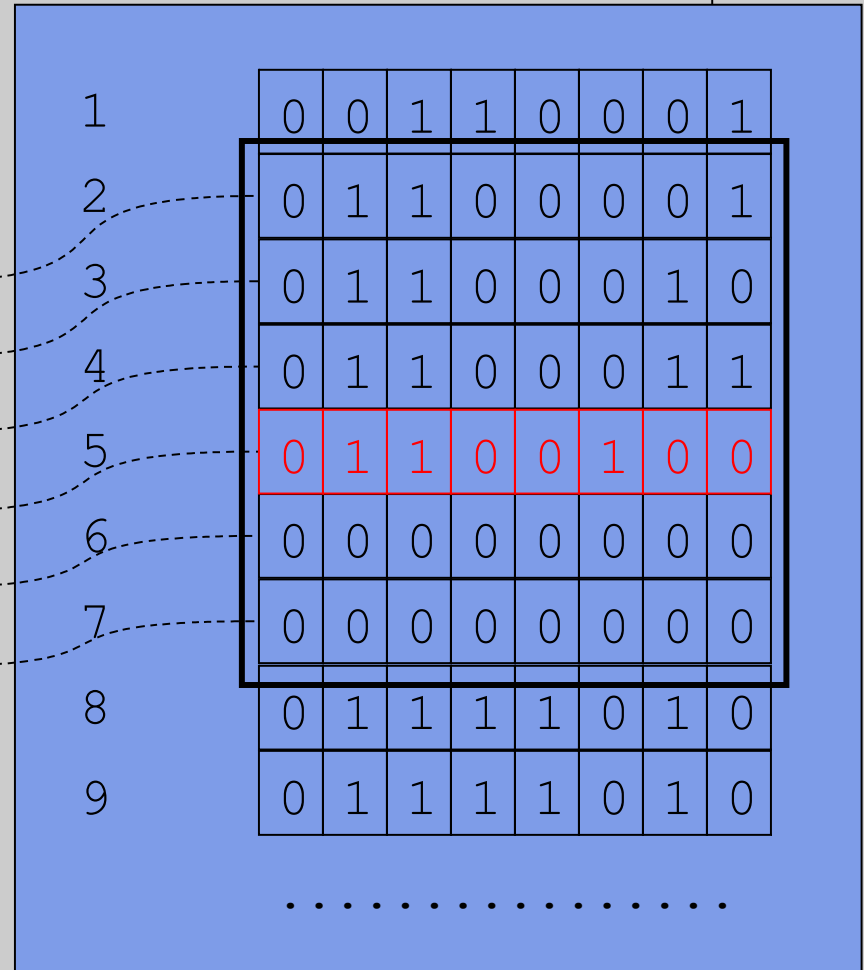




διεύθυνση περιεχόμενα

```
...  
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]

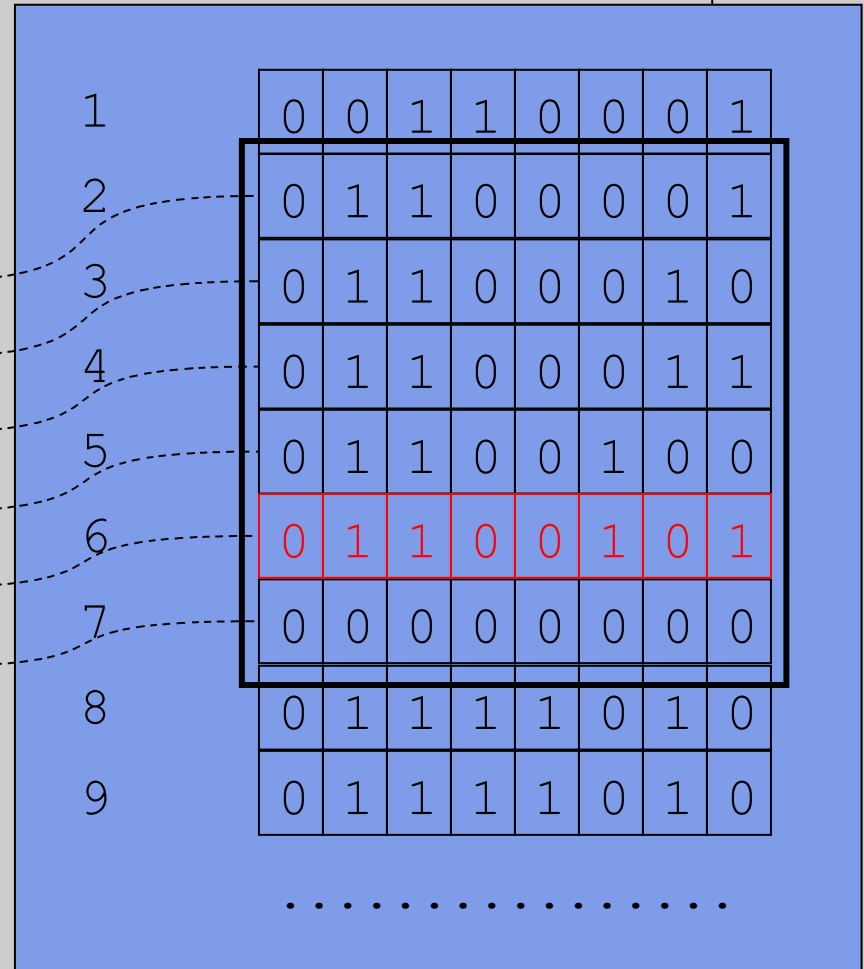




διεύθυνση περιεχόμενα

```
...  
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]

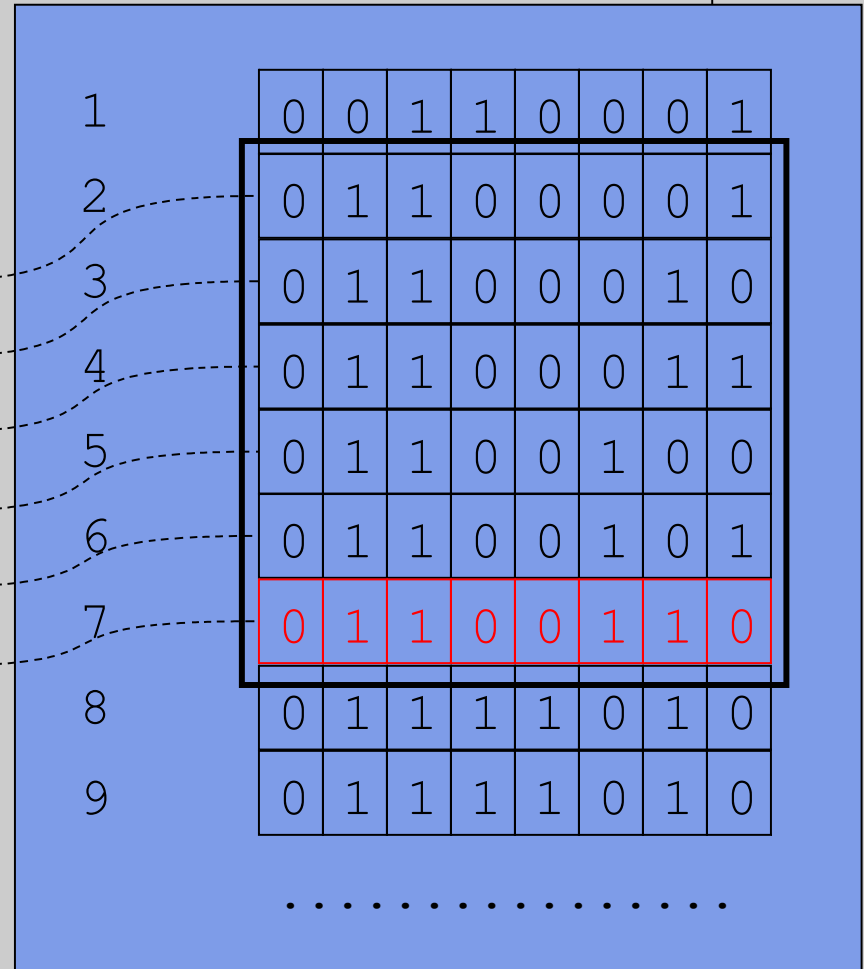




διεύθυνση περιεχόμενα

```
...  
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]

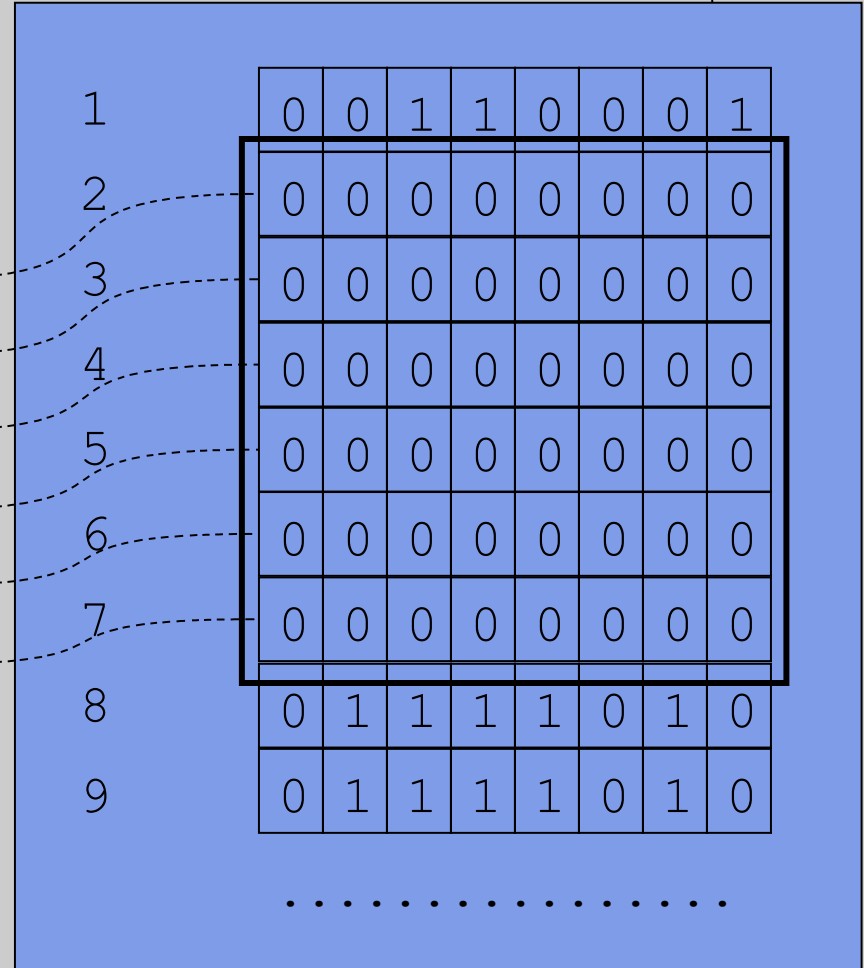




διεύθυνση περιεχόμενα

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

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

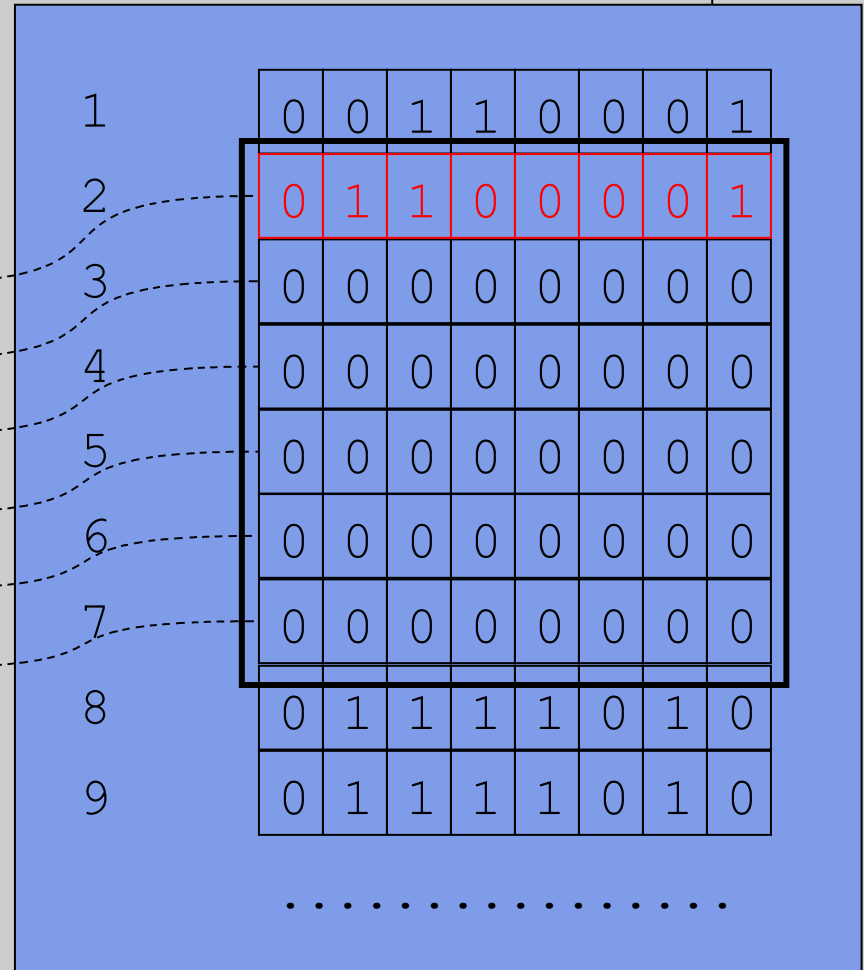




διεύθυνση περιεχόμενα

```
...  
char a[2*3];  
a[0]='a';
```

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

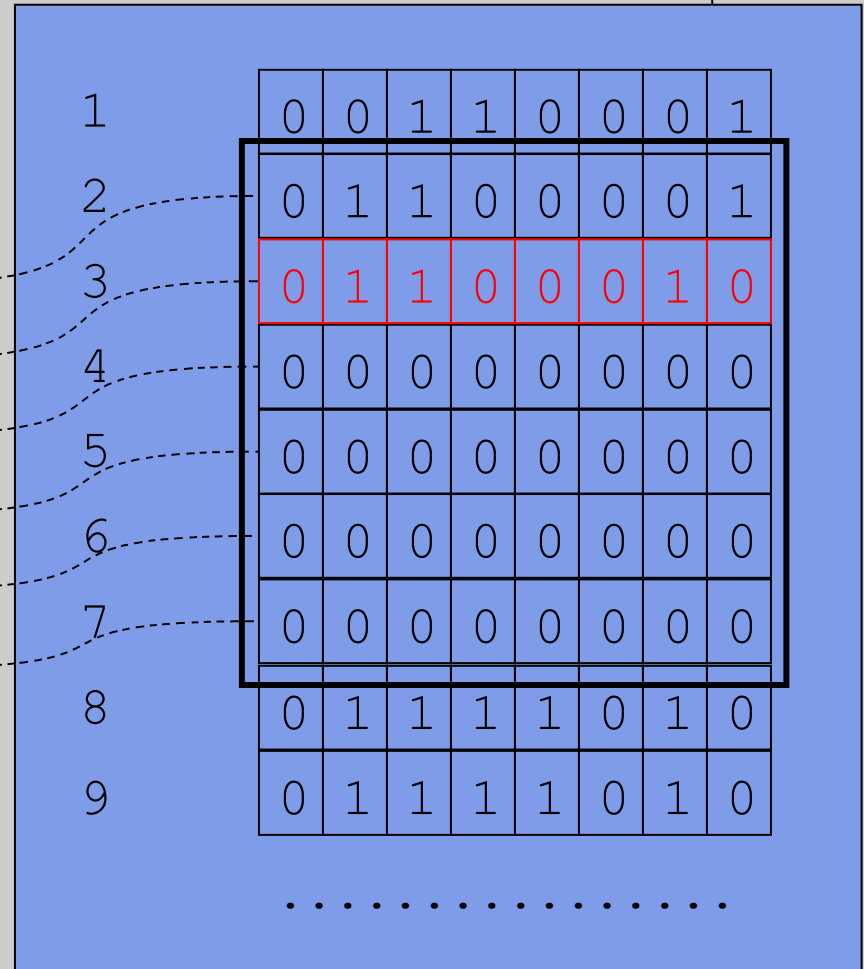




διεύθυνση περιεχόμενα

```
...  
char a[2*3];  
  
a[0]='a';  
  
a[1]='b';
```

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

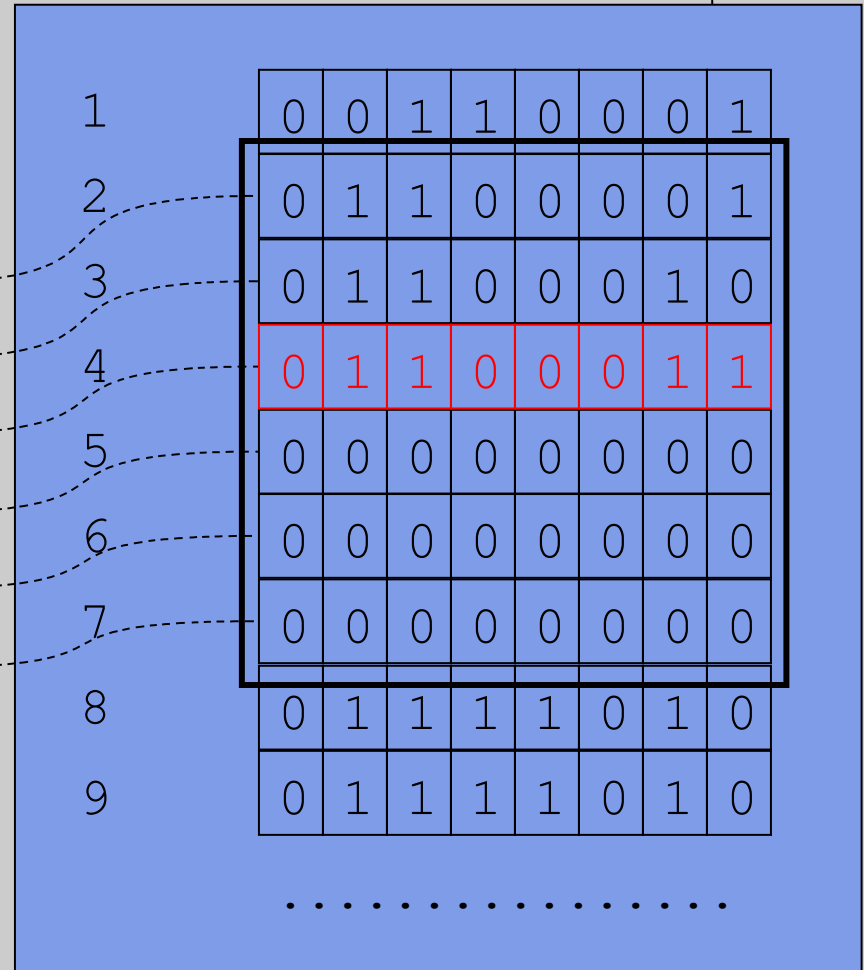




διεύθυνση περιεχόμενα

```
...  
char a[2*3];  
  
a[0]='a';  
  
a[1]='b';  
  
a[2]='c';
```

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

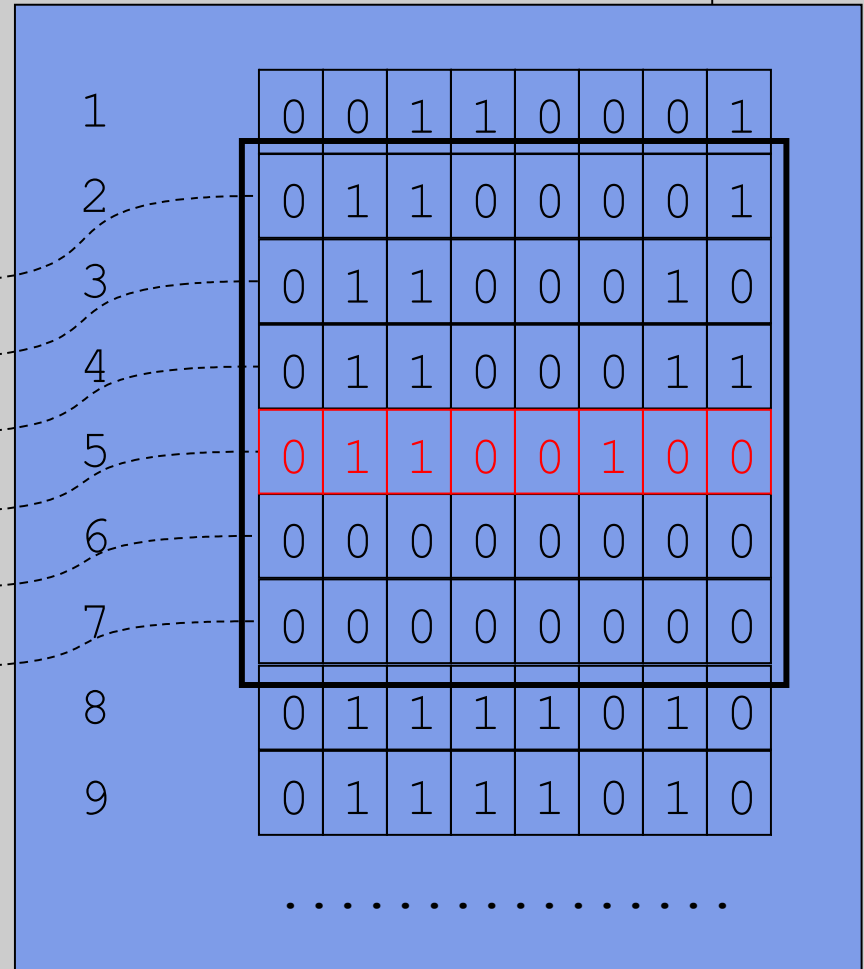




διεύθυνση περιεχόμενα

```
...  
char a[2*3];  
  
a[0]='a';  
a[1]='b';  
a[2]='c';  
a[3]='d';
```

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

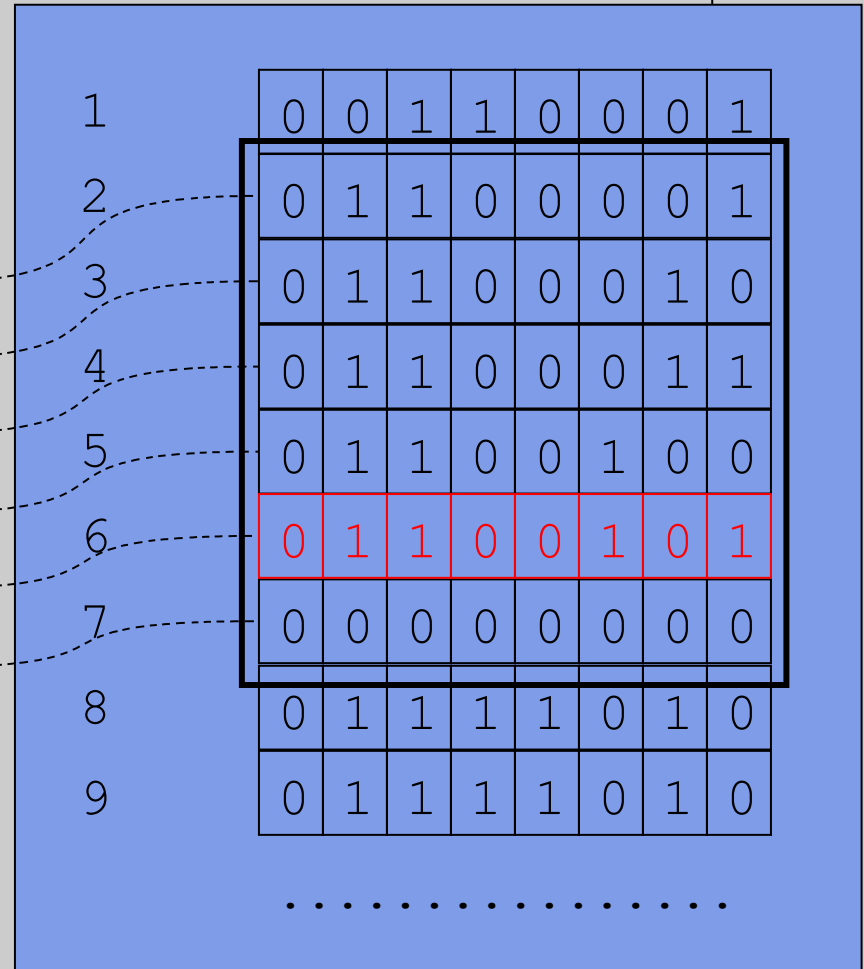




διεύθυνση περιεχόμενα

```
...  
char a[2*3];  
  
a[0]='a';  
a[1]='b';  
a[2]='c';  
a[3]='d';  
a[4]='e';
```

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

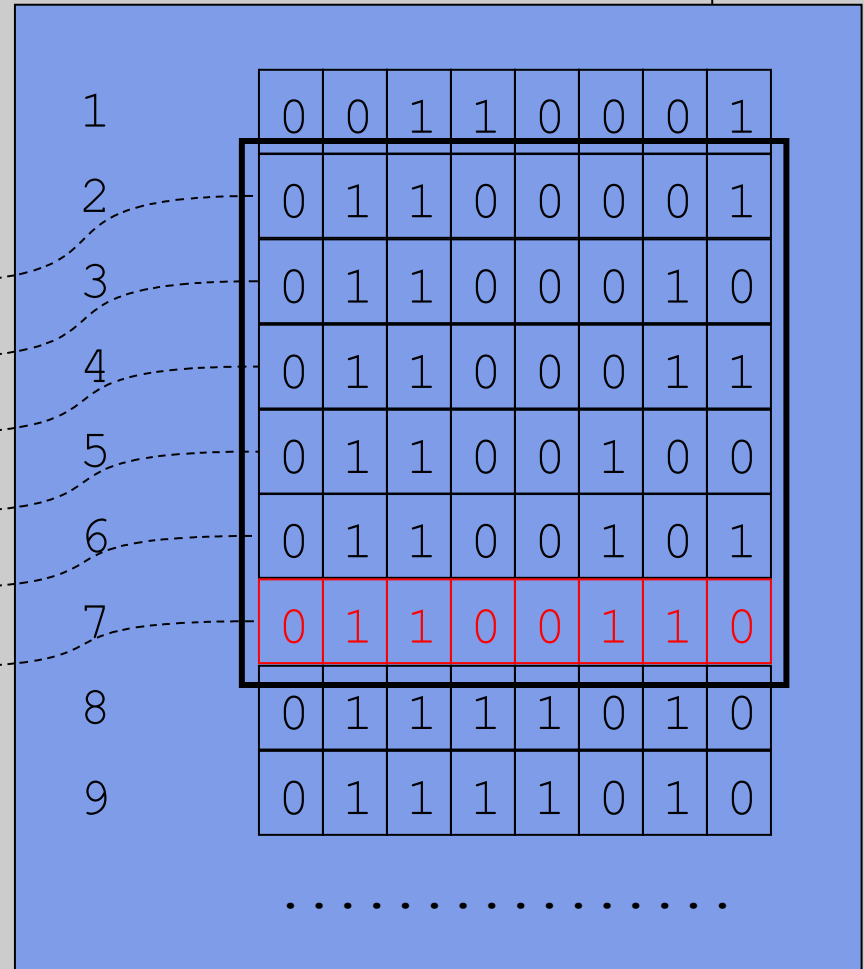




διεύθυνση περιεχόμενα

```
...  
char a[2*3];  
  
a[0]='a';  
a[1]='b';  
a[2]='c';  
a[3]='d';  
a[4]='e';  
a[5]='f';
```

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



Συμβολοσειρές / σειρές χαρακτήρων (strings)



33

- **Strings** υλοποιούνται ως **πίνακες από char**
 - **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 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



35

- Μπορούμε να τα υλοποιήσουμε (εμείς) χαρακτήρα προς χαρακτήρα μέσω της `getchar` και `putchar`.
 - Ή να χρησιμοποιήσουμε την `scanf` και `printf` με τον αντίστοιχο προσδιορισμό, 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:** μπορεί να γίνει αποθήκευση **εκτός ορίων** του πίνακα!
- Αντίθετα με ότι γνωρίζουμε, στην `scanf` **δεν δίνουμε** σαν παράμετρο την **διεύθυνση** της μεταβλητής πίνακα
 - Το γιατί προσεχώς ...



```
#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);
    return(0);
}
```

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

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)



```
/* μήκος string s */  
  
#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 in στο out */

#include <stdio.h>
#define N 32

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

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

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

    /* αντίγραψε ανάποδα στο out */
    for (pos=0, len--; len>=0; pos++, len--) {
        out[pos] = in[len];
    }
    out[pos] = '\0'; /* τερμάτισε το out */

    printf("%s in reverse is %s\n", in, out);
    return(0);
}
```



```
/* αντιγραφή του in1 "και" του in2 στο out */
#include <stdio.h>
#define N 32

int main(int argc, char *argv[]) {
    int i, j;
    char in1[N], in2[N], out[2*N-1];

    printf("enter string:");
    scanf("%31s", in1);
    printf("enter string:");
    scanf("%31s", in2);

    /* αντίγραψε το in1 στο out */
    for (i=0; in1[i]!='\0'; i++) {
        out[i] = in1[i];
    }
```

```
        /* αντίγραψε το in2 στο out */
        for (j=0; in2[j]!='\0'; j++,i++) {
            out[i] = in2[j];
        }
        out[i]='\0'; /* τερμάτισε το out */

    printf("%s + %s = %s\n", in1, in2, out);
    return (0);
}
```

Συναρτήσεις βιβλιοθήκης



40

```
#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);
```


sprintf για τη δημιουργία format string



```
#define SIZE 100
```

```
char word[SIZE];
```

Όταν διαβάζουμε ένα string με scanf πρέπει ΠΑΝΤΑ να προσδιορίζουμε μέγιστο πλήθος χαρακτήρων που επιτρέπεται να αποθηκευτούν σε αυτό

Θέλουμε το scanf να μπορεί να λειτουργήσει σωστά ακόμη κι αν αλλάξει αυτό το πλήθος.

- Άρα, πρέπει να βρούμε τρόπο να κατασκευάσουμε το format string με βάση το επιθυμητό μέγεθος!

sprintf για τη δημιουργία format string



```
#define SIZE 100
```

```
char word[SIZE];
```

θέλουμε να κατασκευάσουμε το
"%99s"

```
#define SIZE 50
```

```
char word[SIZE];
```

θέλουμε να κατασκευάσουμε το
"%49s"

sprintf για τη δημιουργία format string



```
#define SIZE 100  
char word[SIZE];
```

θέλουμε να κατασκευάσουμε το
"%99s"

Από τι αποτελείται?

Το σύμβολο %

```
printf("%%");
```

Ένα ακέραιο με τιμή SIZE-1

```
printf("%d", SIZE-1);
```

Το γράμμα s

```
printf("s");
```