Προγραμματισμός I (ECE115)

#11 πίνακες και δείκτες

Χρήστος Αντωνόπουλος
cda@uth.gr

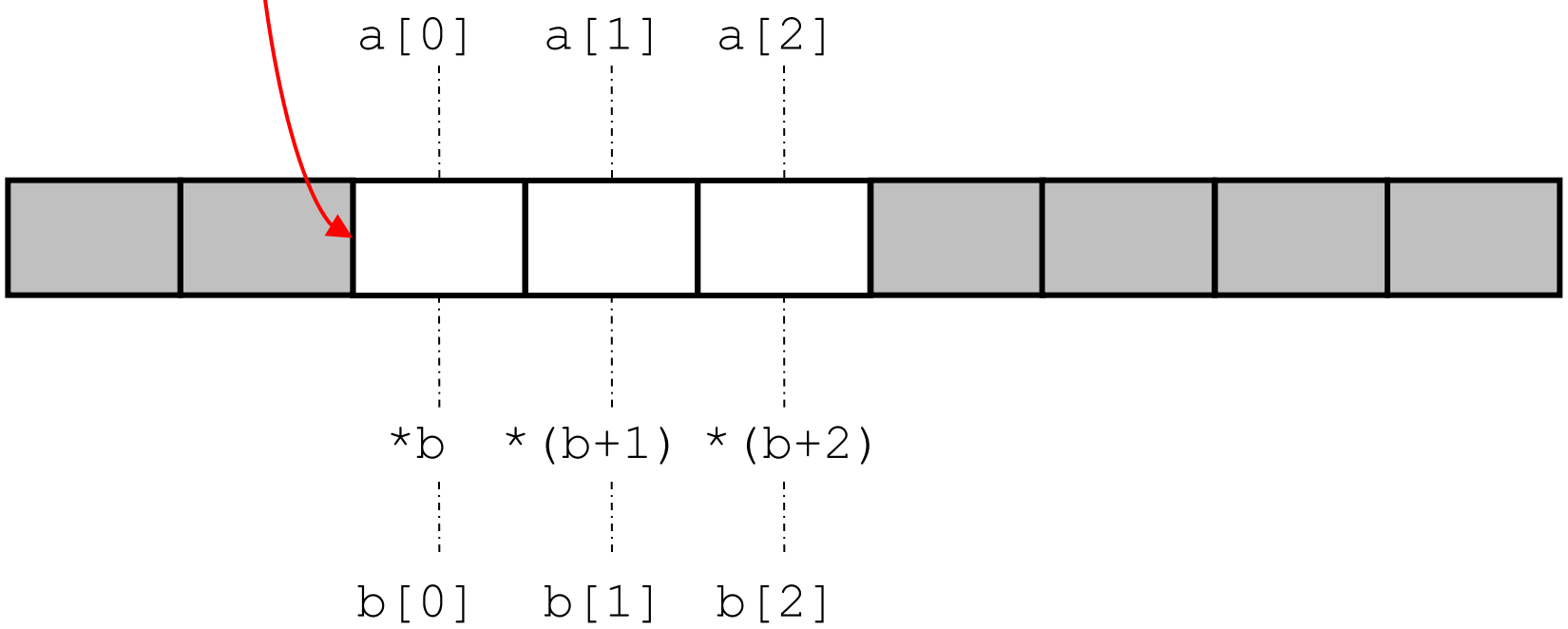
Δείκτες και πίνακες

- Υπάρχει **συντακτική συμβατότητα** ανάμεσα σε `array-of-T` και `pointer-to-T`

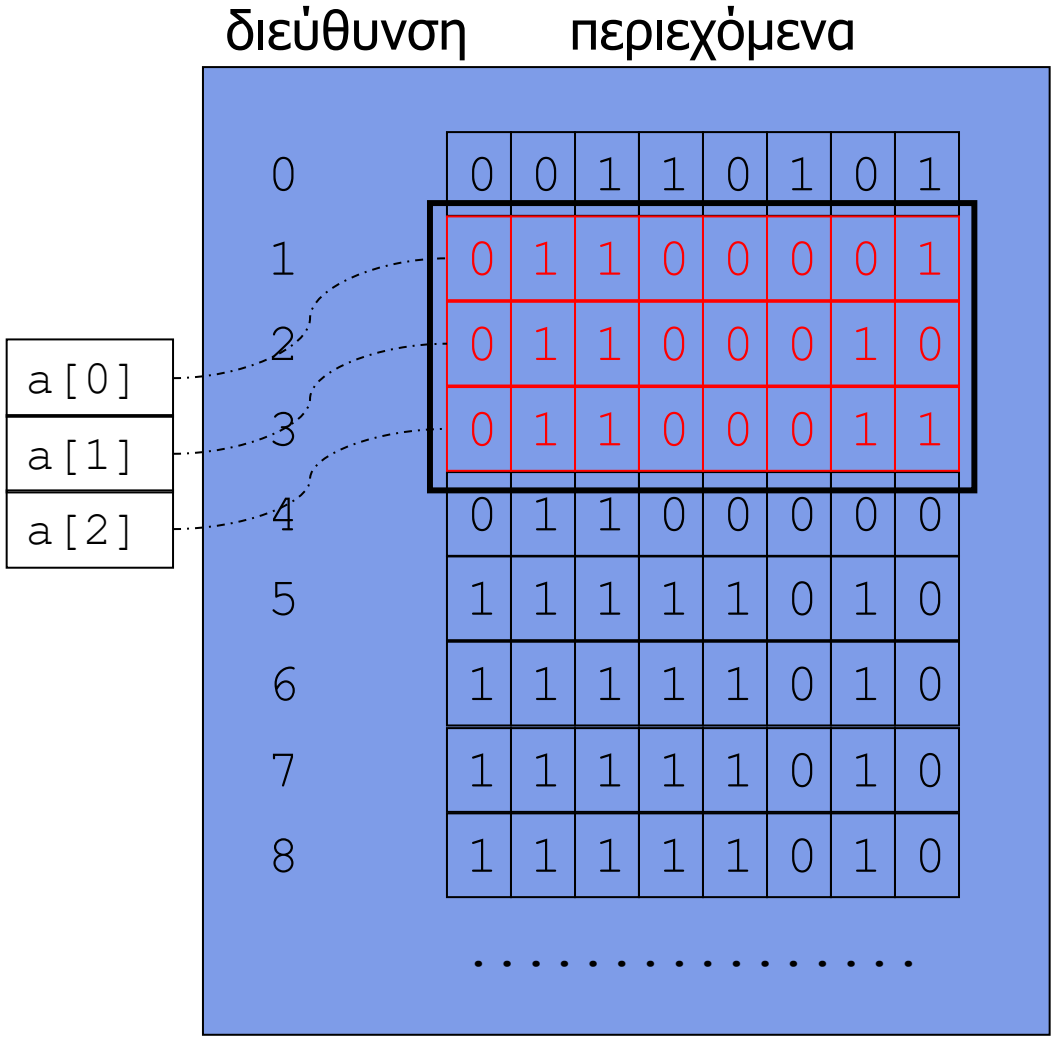
- Μια μεταβλητή `array-of-T` μπορεί να θεωρηθεί ως μια μεταβλητή `pointer-to-T` (με τιμή την διεύθυνση του πρώτου στοιχείου του πίνακα)
- Μια μεταβλητή `pointer-to-T` μπορεί να θεωρηθεί ως η αρχή ενός `array-of-T`

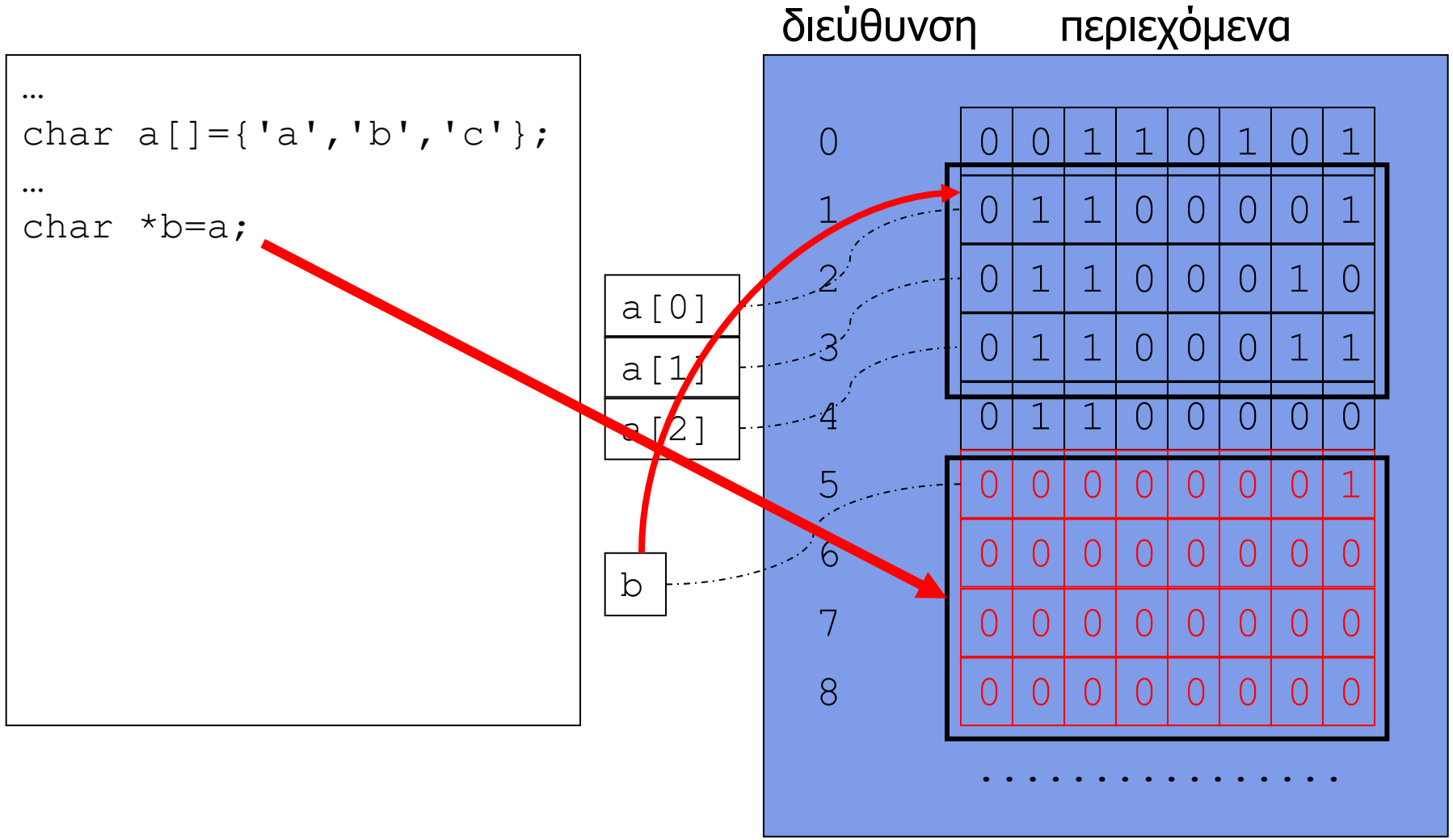
- Με χρήση δεικτών μπορεί να γίνει **διέλευση** των στοιχείων ενός πίνακα
  - αντί της συμβατικής πρόσβασης μέσω θέσης στον πίνακα

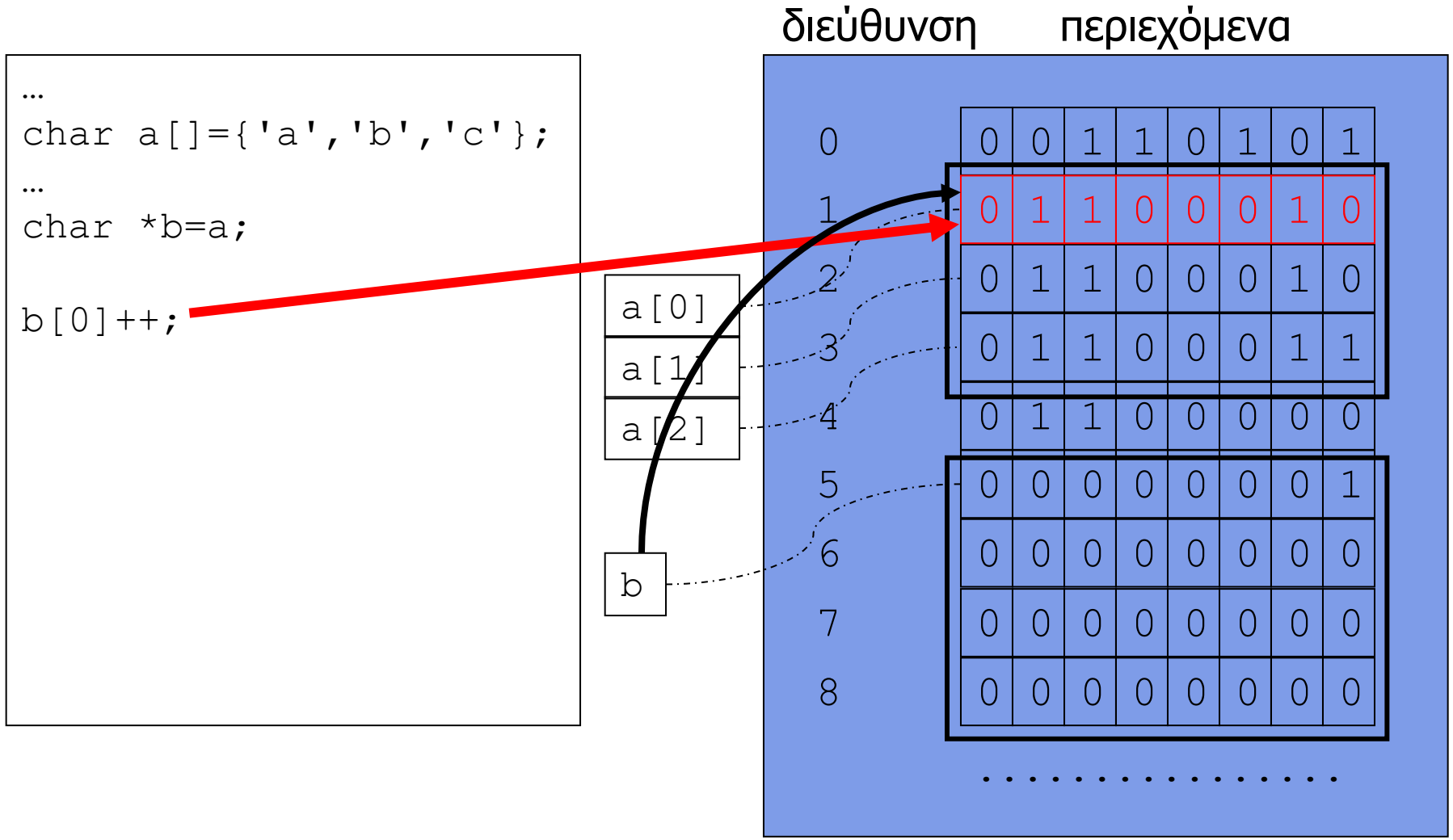
```
T a[3];
T *b = a;
```

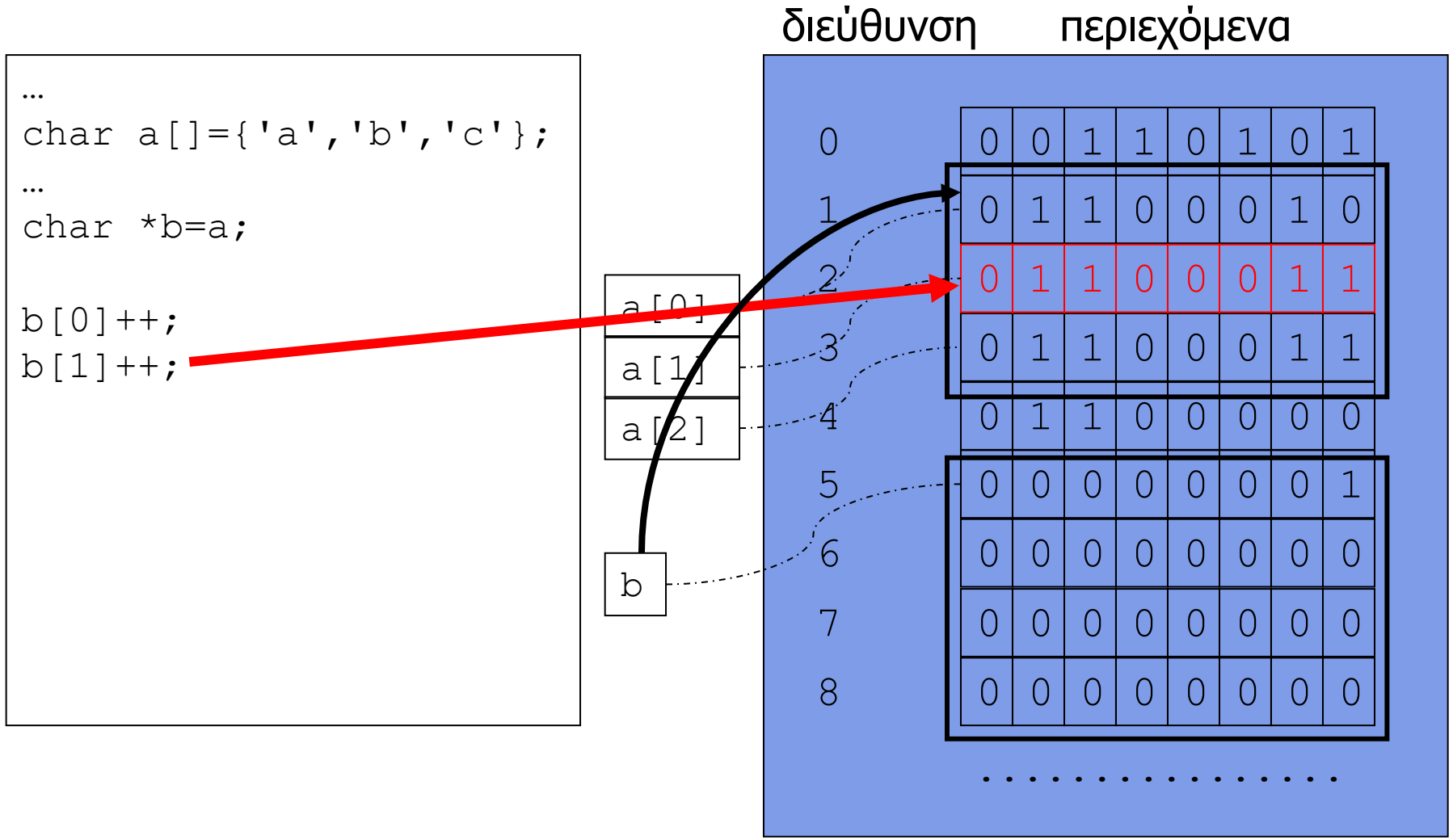


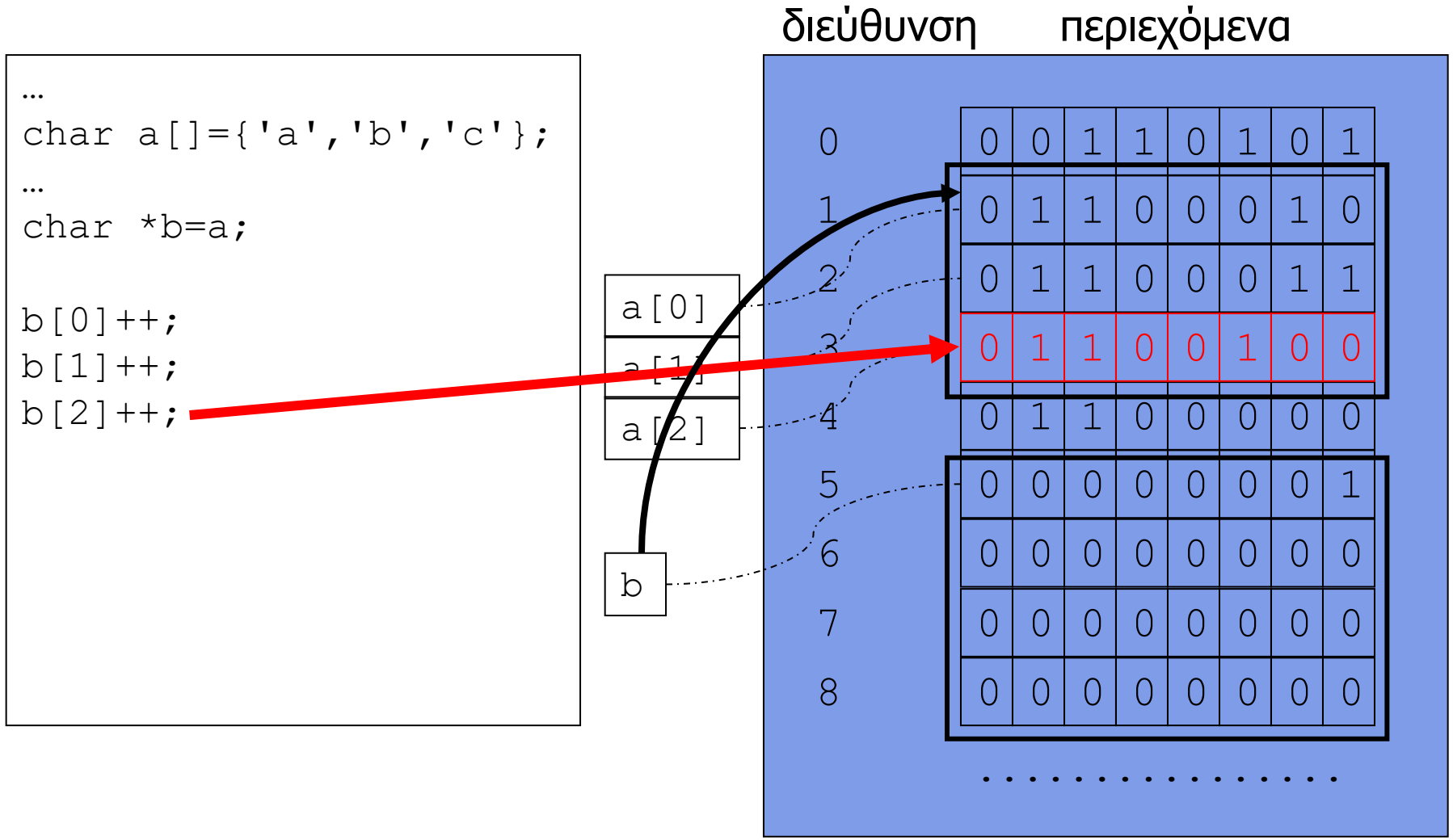
```
…
char a[]={'a','b','c'};
…
```

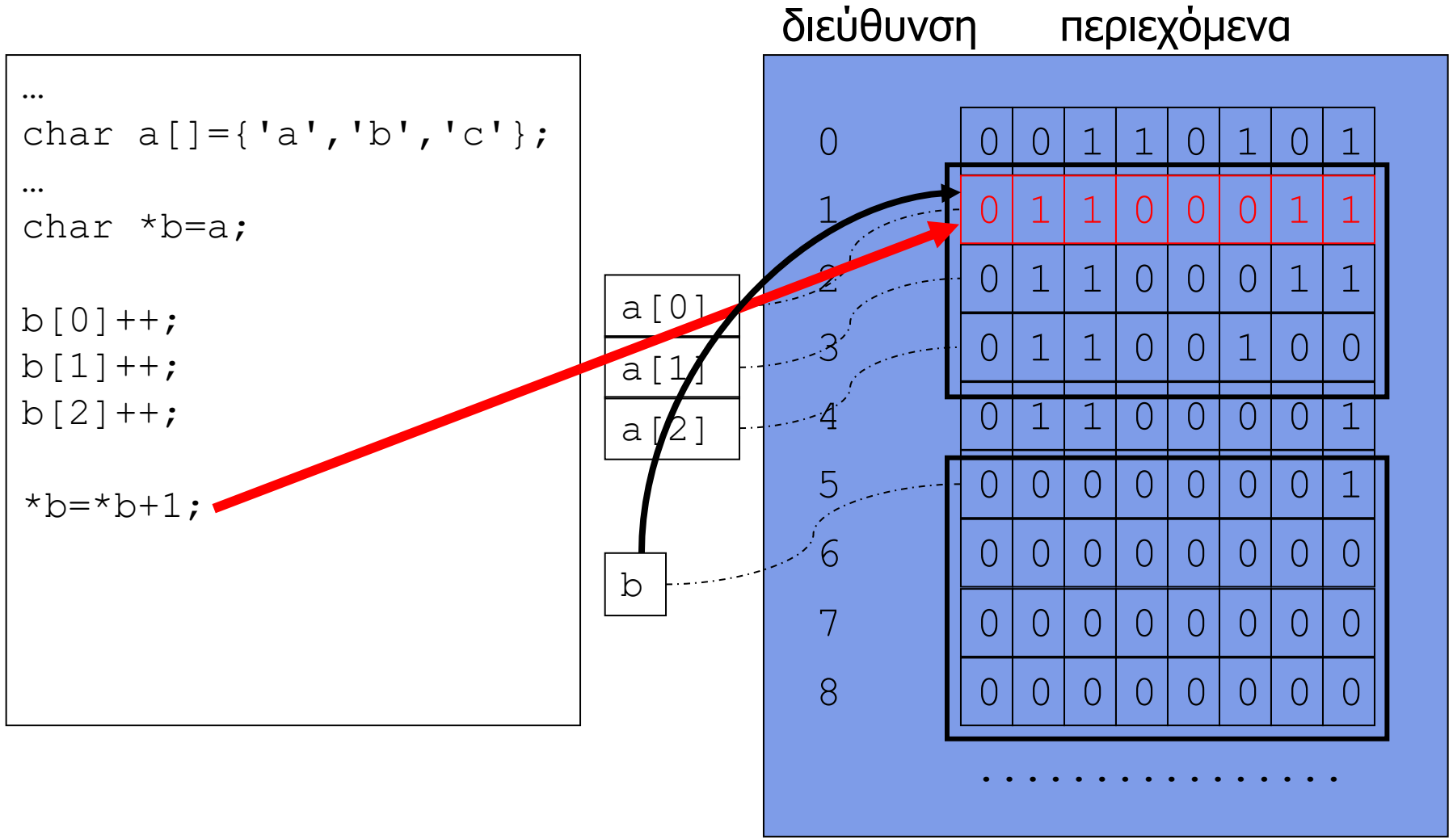

διεύθυνση
περιεχόμενα
a[0]
a[1]
a[2]
0 0 1 1 0 1 0 1
1 0 1 1 0 0 0 0 1
2 0 1 1 0 0 0 1 0
3 0 1 1 0 0 0 1 1
4 0 1 1 0 0 0 0 0
5 1 1 1 1 1 0 1 0
6 1 1 1 1 1 0 1 0
7 1 1 1 1 1 0 1 0
8 1 1 1 1 1 0 1 0
. . . . . .

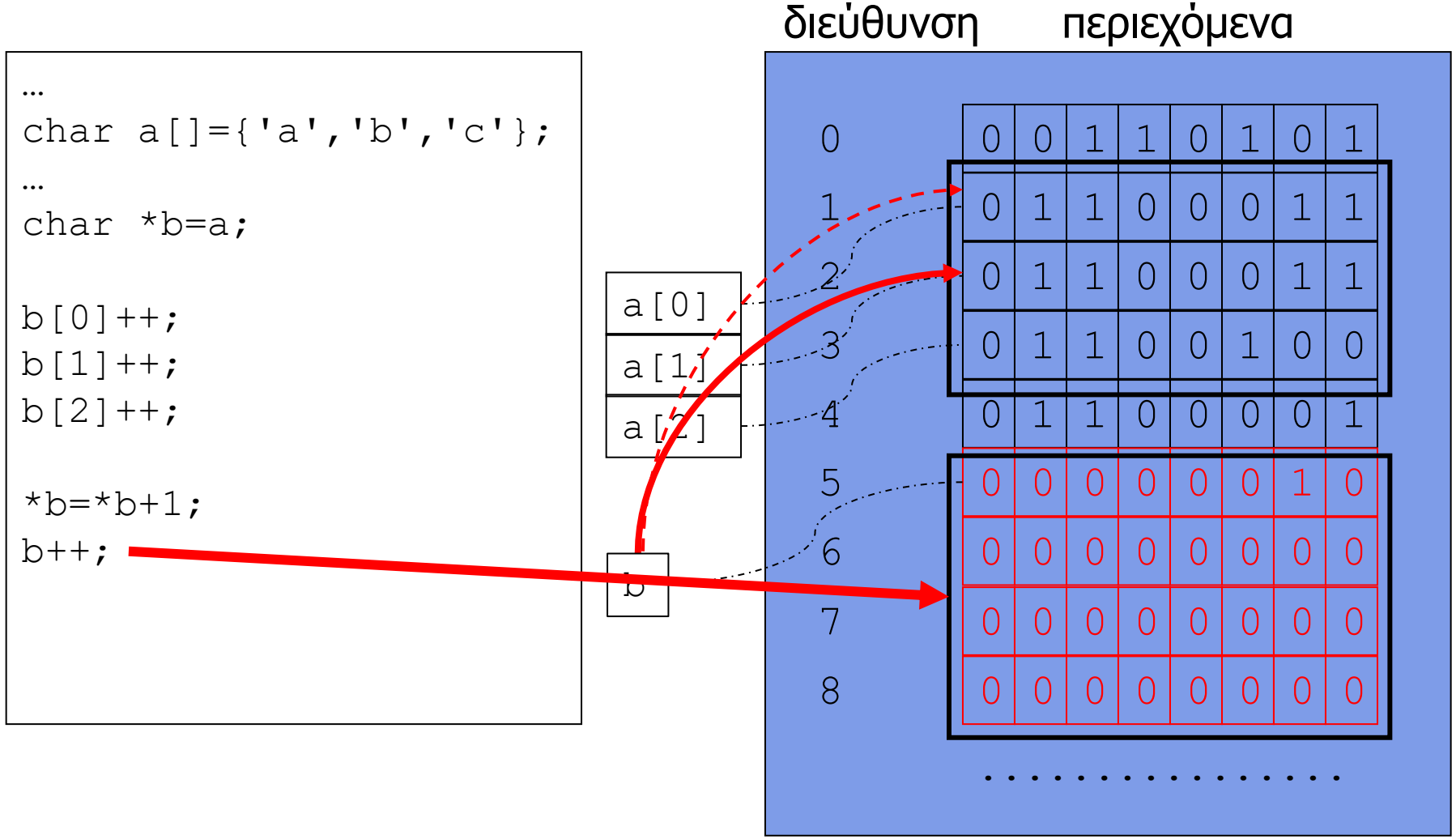

διεύθυνση
περιεχόμενα
…
char a[]={'a','b','c'};
…
char *b=a;
a[0]
a[1]
a[2]
b
0
1
2
3
4
5
6
7
8
0 0 1 1 0 1 0 1
0 1 1 0 0 0 0 1
0 1 1 0 0 0 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

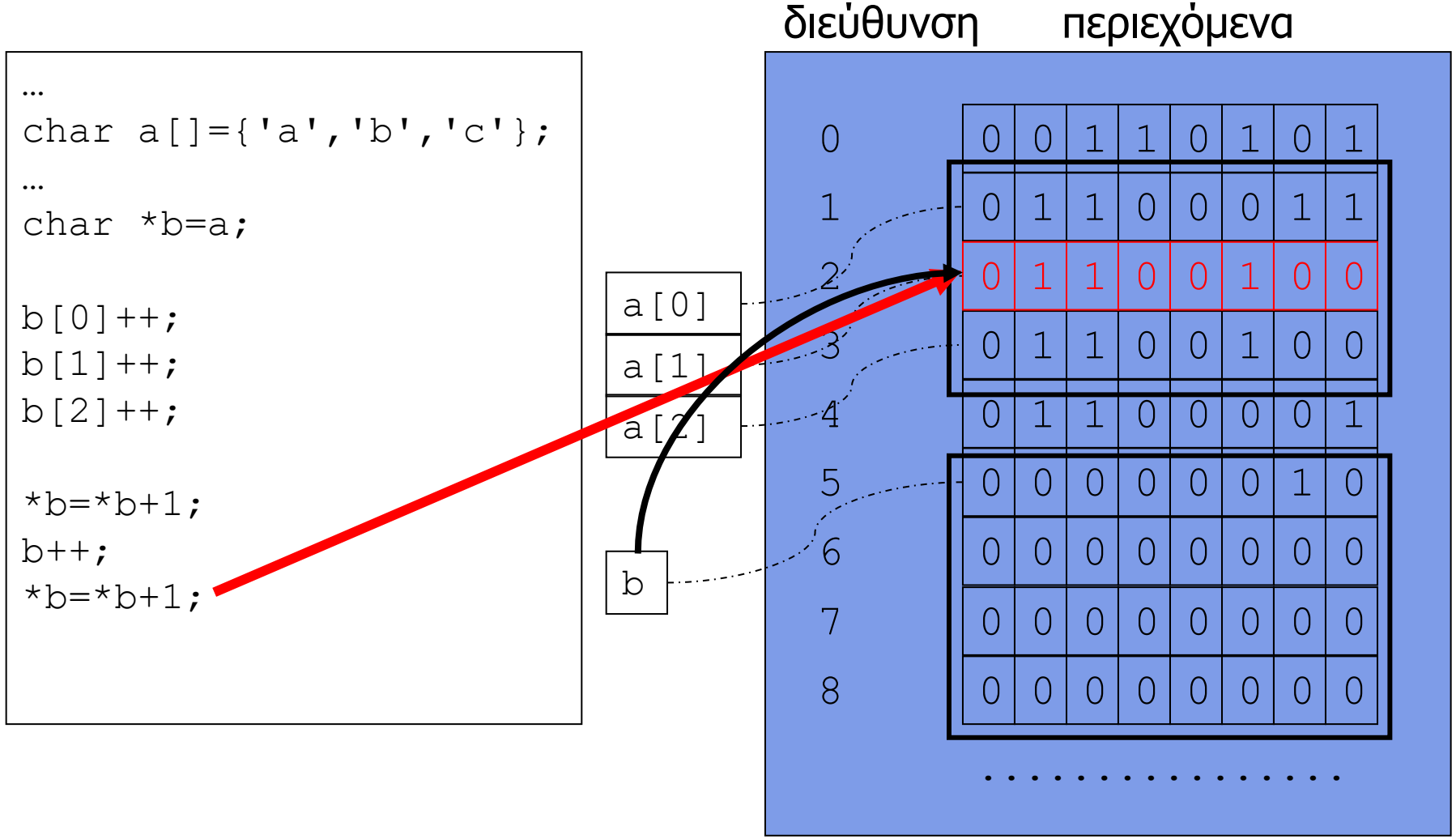

διεύθυνση
περιεχόμενα
…
char a[]={'a','b','c'};
…
char *b=a;
b[0]++;
a[0]
a[1]
a[2]
b
0
1
2
3
4
5
6
7
8
0 0 1 1 0 1 0 1
0 1 1 0 0 0 1 0
0 1 1 0 0 0 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

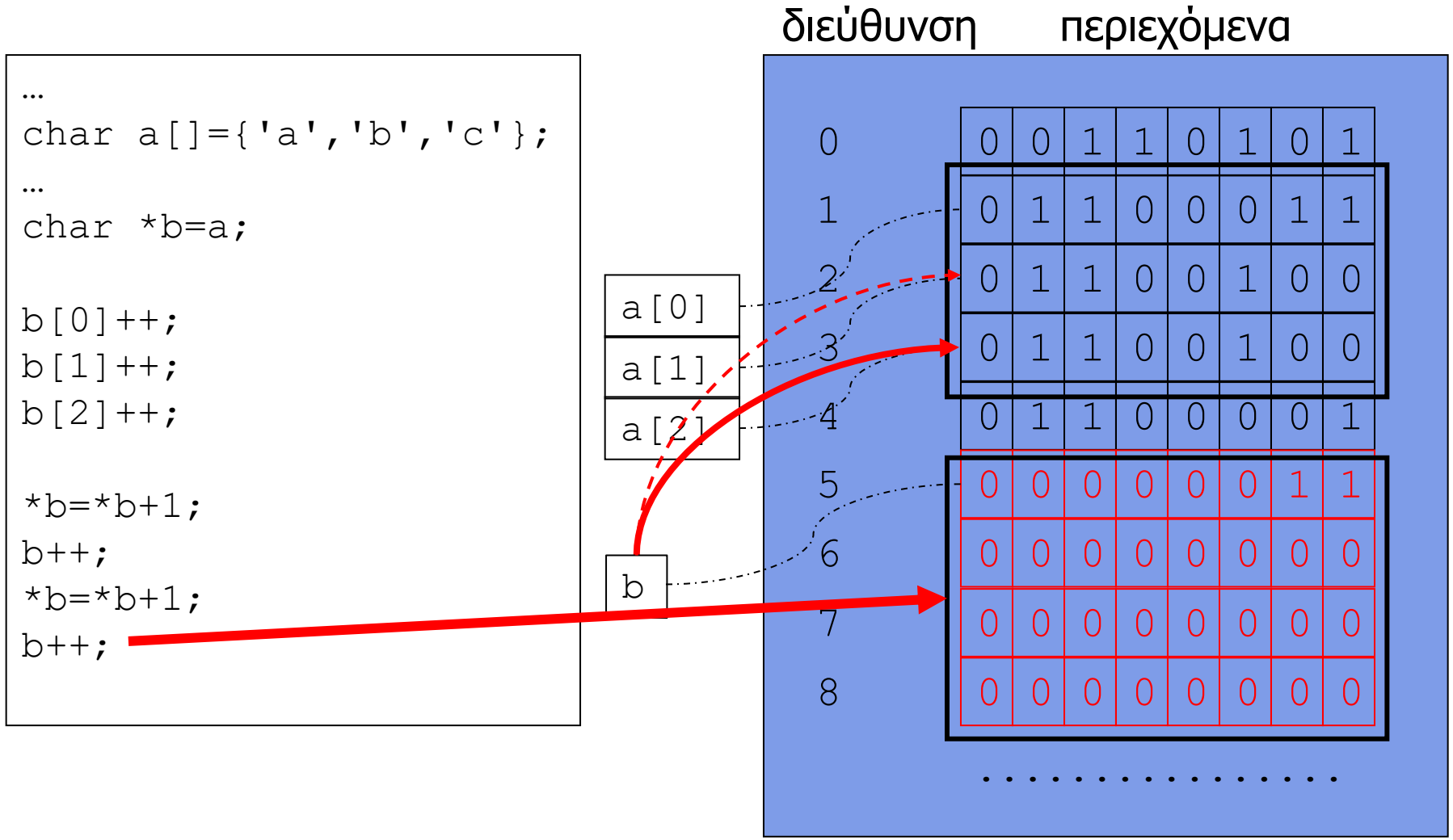

διεύθυνση
περιεχόμενα
…
char a[]={'a','b','c'};
…
char *b=a;
b[0]++;
b[1]++;
a[0]
a[1]
a[2]
b
0
1
2
3
4
5
6
7
8
0 0 1 1 0 1 0 1
0 1 1 0 0 0 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 0 1 1
0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

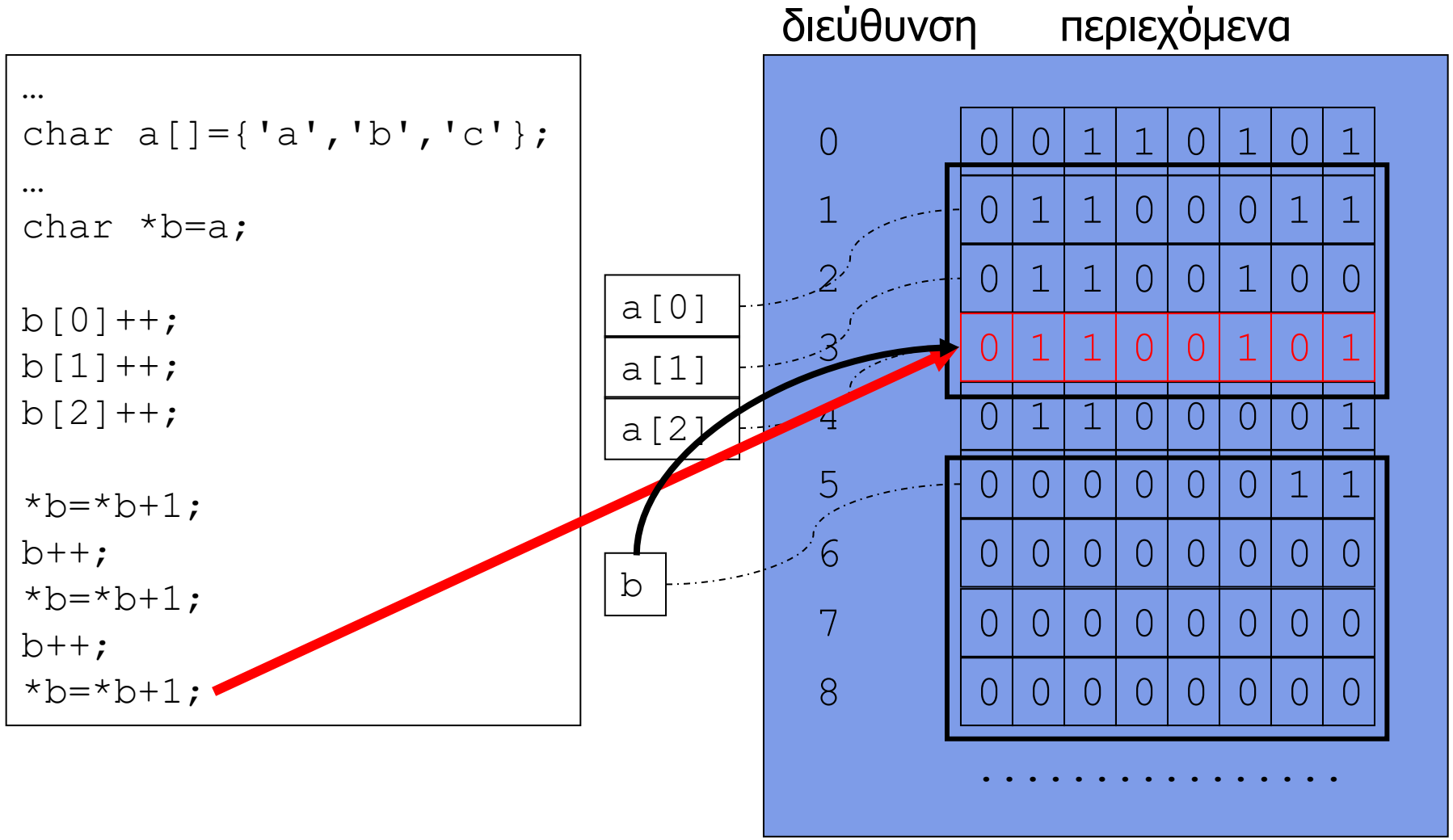

διεύθυνση
περιεχόμενα
…
char a[]={'a','b','c'};
…
char *b=a;
b[0]++;
b[1]++;
b[2]++;
a[0]
a[1]
a[2]
b
0
1
2
3
4
5
6
7
8
0 0 1 1 0 1 0 1
0 1 1 0 0 0 1 0
0 1 1 0 0 0 1 1
0 1 1 0 0 1 0 0
0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0


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


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


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


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


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

Προσπέλαση πίνακα με δείκτες

- Η πρόσβαση στα στοιχεία ενός πίνακα μέσω δείκτη μπορεί να είναι **πιο γρήγορη** από την συμβατική πρόσβαση μέσω θέσης στον πίνακα
  - γιατί;
- Όμως, **δεν** είναι καλή ιδέα να χρησιμοποιείται
  - εκτός αν υπάρχει σοβαρός λόγος, π.χ. η συγκεκριμένη πρόσβαση αποτελεί σημείο συμφόρησης κρίσιμου κώδικα
- Η συμβατική πρόσβαση στα στοιχεία ενός πίνακα συνήθως **βελτιώνει** την αναγνωσιμότητα του κώδικα
- Η πρόσβαση με δείκτες είναι συνήθης τακτική κυρίως για προγραμματισμό σε χαμηλό επίπεδο συστήματος
  - π.χ. μέσα στο ίδιο το λειτουργικό σύστημα

```
int a[N], i;

for (i=0; i<N; i++)
  printf("%d ", a[i]);
```

```
int a[N], *p;

for (p=a; p<a+N; p++)
  printf("%d ", *p);
```

```
char s[]= "onetwothree";
char *sptr;

printf("%s\n", s);

sptr = s;

printf("%s\n", sptr);

printf("%s\n", &s[3]);

printf("%s\n", sptr + 6);
```

Δείκτες αντί για πίνακες ως παράμετροι συνάρτησης καθ' αναφορά

- Οι πίνακες περνιούνται ως παράμετροι στις συναρτήσεις πάντα καθ' αναφορά
  - διεύθυνση στο πρώτο στοιχείο του πίνακα

- Η αντίστοιχη τυπική παράμετρος της συνάρτησης μπορεί να δηλωθεί **ως δείκτης** (αντί για πίνακας)
  - ο κώδικας της συνάρτησης εξακολουθεί να μπορεί να προσπελάσει τις θέσεις μνήμης σαν να ήταν πίνακας
  - εναλλακτικά, μέσω δείκτη

```c
#include<stdio.h>

#define SIZE 5

double calcAverage(int *vals, int size) {
  int i;
  double sum = 0;

  for (i = 0; i < size; ++i) {
    sum = sum + vals[i];
  }
  return(sum / size);
}

int main () {
  int v[SIZE] = {1000, 2, 3, 17, 50};
  double avg;

  avg = calcAverage(v, SIZE) ;
  printf("Average value is: %lf\n", avg);

  return(0);
}
```

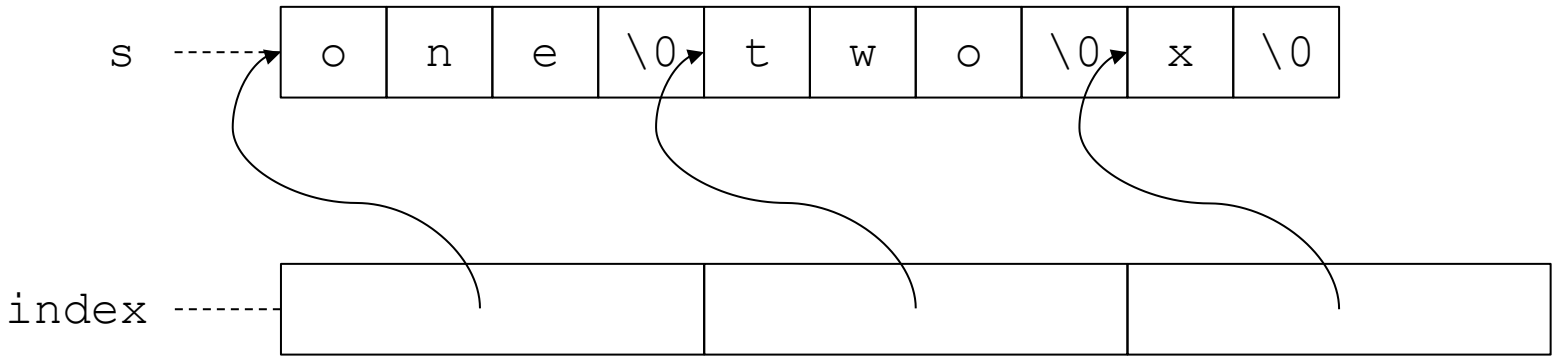
```c
#include <stdio.h>

#define N 16

void smallToCapitals(char *s) {
  int i;

  for (i=0; s[i] != '\0'; i++) {
    if ( (s[i] >= 'a') && (s[i] <= 'z') ) {
      s[i] = 'A' + s[i] - 'a';
    }
  }
}

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

  scanf("%15s",str);
  smallToCapitals(str);
  printf("%s\n",str);

  return(0);
}
```

Πίνακας από δείκτες

- Μπορούμε να φτιάξουμε **πίνακες από δείκτες**
- Τα στοιχεία του πίνακα προσπελάζονται ως συνήθως
  - με βάση τους κανόνες που ισχύουν για τους πίνακες
- Απλά, κάθε στοιχείο του πίνακα είναι ένας δείκτης

```
int *iptrs[];    /* πίνακας pointer-to-int */


itprs[3] = … ;   /* γράψιμο του 4ου στοιχείου */
*iptrs[3] = … ;  /* γράψιμο στην διεύθυνση μνήμης
                    όπου δείχνει το 4ο στοιχείο */
… = iptrs[3];    /* ανάγνωση του 4ου στοιχείου */
… = *iptrs[3];   /* ανάγνωση από την διεύθυνση μνήμης
                    όπου δείχνει το 4ο στοιχείο */
```

```
/* εκτύπωση strings αποθηκευμένων σε ένα πίνακα  */

char s[]= {'o','n','e','\0','t','w','o','\0','x','\0'};
char *index[3];

index[0] = &s[0];
index[1] = &s[4];
index[2] = &s[8];

printf("%s %s %s\n", index[0], index[1], index[2]);
```

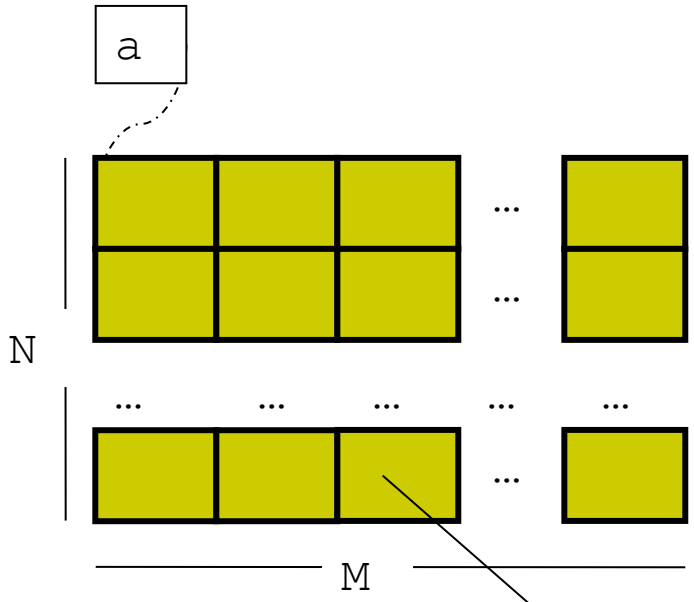


Πίνακες από δείκτες & 2-διάστατοι πίνακες

- Οι πίνακες από δείκτες (π.χ. `*b[N]`) **μοιάζουν** με 2-διάστατους πίνακες (π.χ. `a[N][M]`), **όμως** …
- Οι γραμμές του πίνακα `a` αποθηκεύονται σε συνεχόμενες διευθύνσεις / θέσεις στην μνήμη
- Ενώ κάθε δείκτης `b[i]` του πίνακα `b` μπορεί να δείχνει σε εντελώς **διαφορετική** περιοχή μνήμης
  - που δεν έχει καμία σχέση με τις περιοχές μνήμης όπου δείχνουν οι υπόλοιποι δείκτες του πίνακα `b`
- Κάθε γραμμή `a[i]` του 2-διάστατου πίνακα `a` έχει **ακριβώς** τον ίδιο αριθμό στοιχείων (`M`)
- Ενώ ένας δείκτης `b[i]` του πίνακα `b` μπορεί να μην δείχνει πουθενά (να είναι `NULL`) ή να δείχνει σε μια σειρά από διαφορετικό αριθμό αντικειμένων

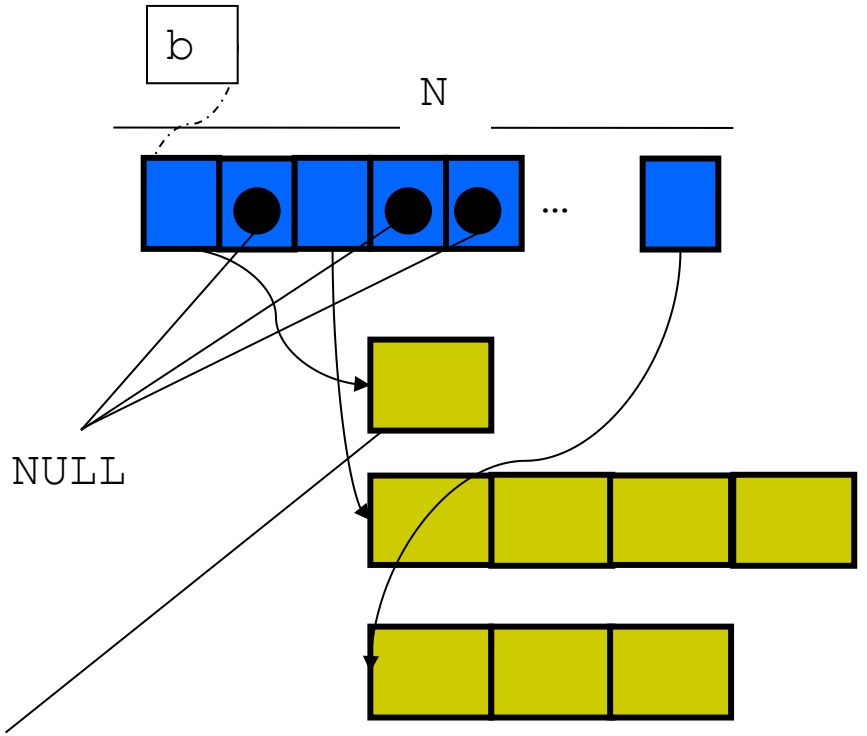
2-διάστατος πίνακας αντικειμένων τύπου `T`

```
T a[N][M];
```



πίνακας από δείκτες σε αντικείμενα τύπου `T`

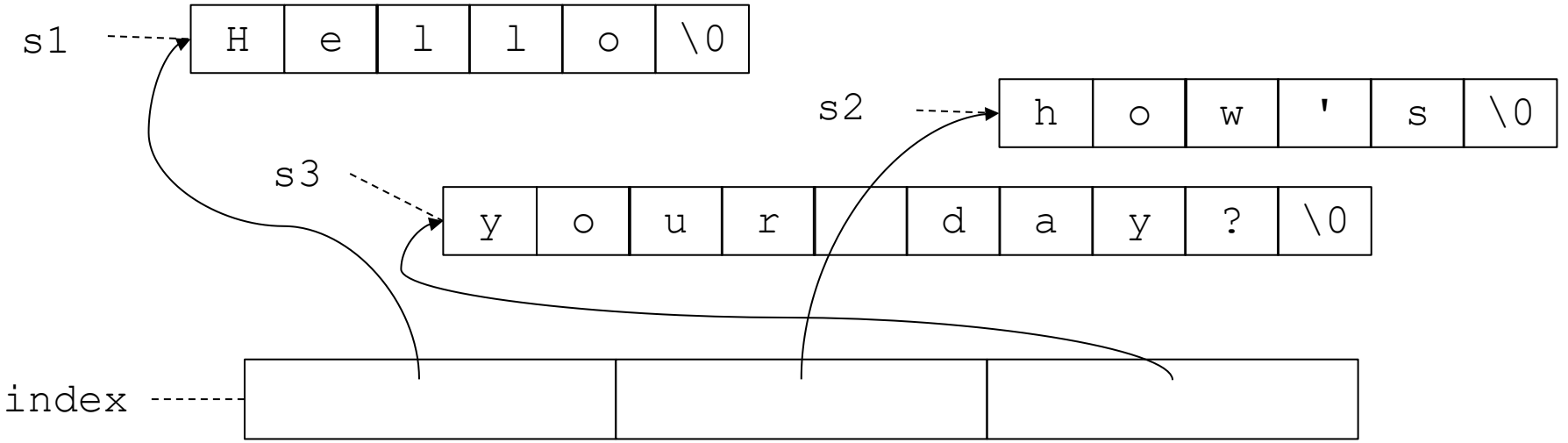
```
T *b[N];
```



δεδομένα τύπου `T`

```
char s[3][6]= {"one", "two", "three"};
char *sptr;

printf("%s\n", s[0]);
printf("%s\n", s[1]);
printf("%s\n", s[2]);

sptr = s[0];
printf("%s\n", sptr);

sptr = s[1];
printf("%s\n", sptr);

sptr = s[2];
printf("%s\n", sptr);
```

```
char s1[] = "Hello";
char s2[] = "how's";
char s3[] = "your day?";

char *index[3];

index[0] = s1;
index[1] = s2;
index[2] = s3;

printf("%s %s %s\n", index[0], index[1], index[2]);
```



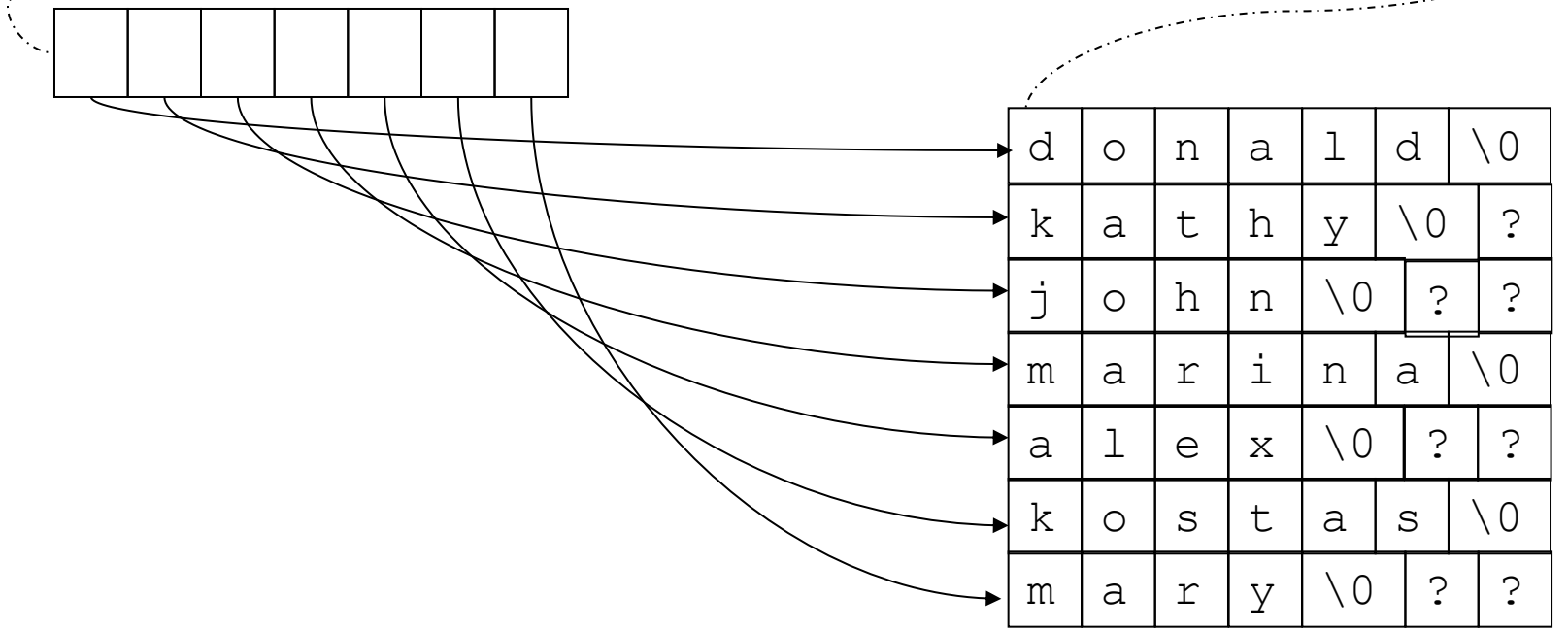
Χρήση ευρετηρίου
Ευρετήριο: **πριν** την ταξινόμηση

πίνακας από 7 δείκτες
σε ονόματα στον πίνακα

```
char *idx[7];
```

πίνακας από 7 ονόματα
(το πολύ 6 χαρακτήρων)

```
char names[7][7];
```



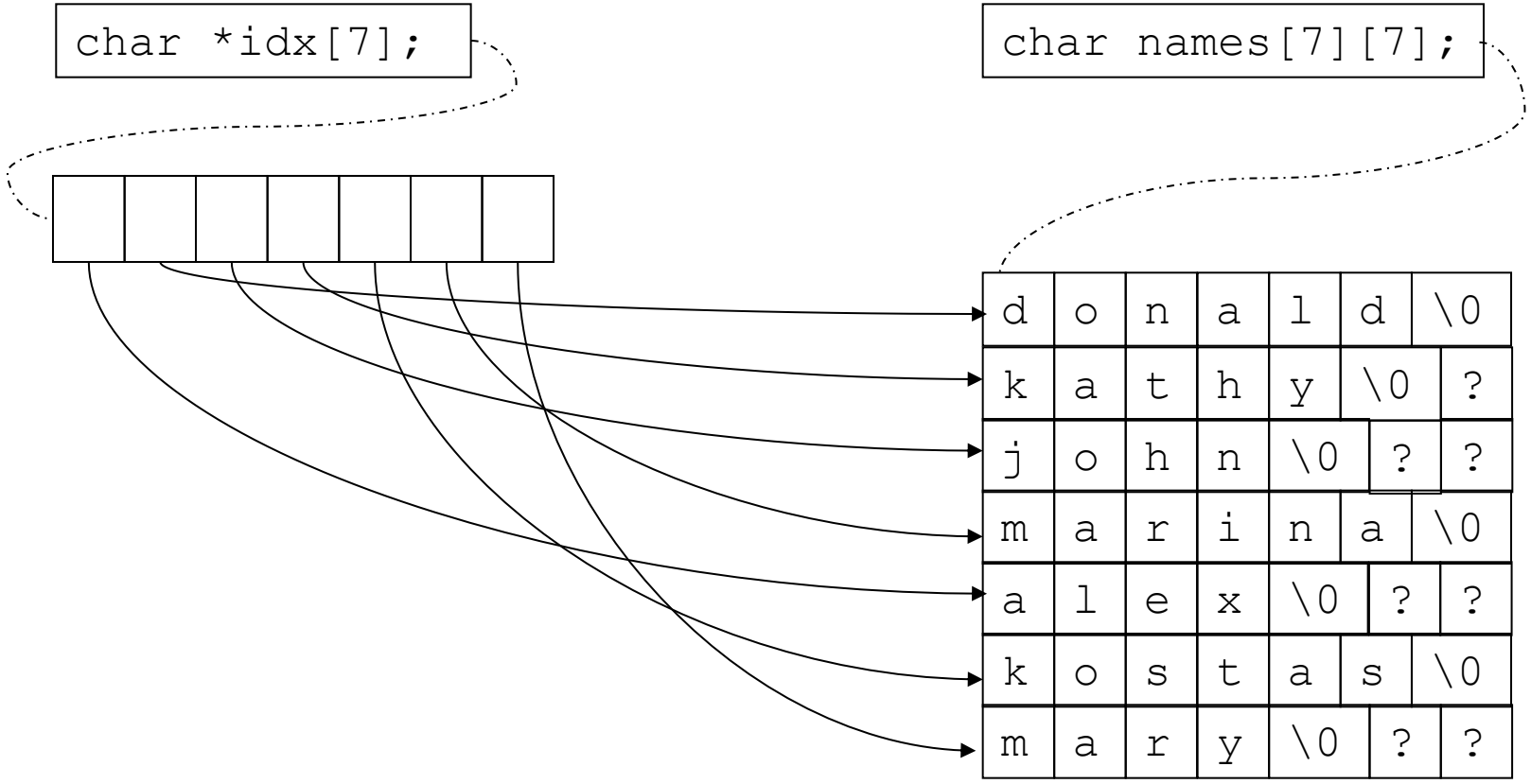
Ευρετήριο: **μετά** την ταξινόμηση

πίνακας από 7 δείκτες
σε ονόματα στον πίνακα

```
char *idx[7];
```

πίνακας από 7 ονόματα
(το πολύ 6 χαρακτήρων)

```
char names[7][7];
```



```c
/* ταξινόμηση αλφαριθμητικών με ευρετήριο */
#include <stdio.h>
#include <string.h>
#define N 7
#define MAXNAMELEN 7

int main(int argc, char *argv[]) {
  char names[N][MAXNAMELEN],*idx[N],*tmp; int i,j;

  for (i=0; i<N; i++) {
    printf("enter name: "); scanf("%6s", names[i]);
  }

  for (i=0; i<N; i++) { idx[i] = names[i]; }

  for (i=0; i<N; i++) {
    for (j=i; j<N; j++) {
      if (strcmp(idx[i],idx[j])>0) {
        tmp = idx[i]; idx[i] = idx[j]; idx[j] = tmp;
      }
    }
  }
  for(i=0; i<N; i++) { printf("%s\n", idx[i]); }
  return(0);
}
```

Εισαγωγή strings με περιορισμό χαρακτήρων

snprintf

```
int snprintf(char *str, size_t size,
             const char *format, ...);
```

- λειτουργεί όπως η `sprint`
- στο `str` γράφονται **το πολύ** `size` χαρακτήρες
- συμπεριλαμβανομένου **και** του τερματικού

Ανάγνωση string με περιορισμό χαρακτήρων

- Όταν διαβάζουμε ένα string με `scanf` πρέπει **πάντα** να προσδιορίζουμε μέγιστο πλήθος χαρακτήρων που επιτρέπεται να αποθηκευτούν στον αντίστοιχο πίνακα
- Ιδανικά, θέλουμε όχι μόνο το μέγεθος του πίνακα αλλά **και** ο μέγιστος αριθμός χαρακτήρων που διαβάζονται μέσω `scanf` να ορίζεται ως **συμβολική σταθερά**
- Όμως, μέχρι στιγμής το δεύτερο γίνεται με το χέρι …

```
#define SIZE 32

char str[SIZE];

scanf("%31s",str);
```

αυτό θέλουμε να βγαίνει αυτόματα

Χρήση `snprintf` για αυτοματοποιημένη δημιουργία format string για την `scanf`

```
#define SIZE 32

char str[SIZE];
```

θέλουμε να κατασκευάσουμε το format string "`%31s`"

από τι αποτελείται;

- το σύμβολο `%` → `printf("%%");`
- ένα ακέραιο με τιμή `SIZE-1` → `printf("%d", SIZE-1);`
- το γράμμα `s` → `printf("s");`

- Χρησιμοποιούμε την `snprintf` για να εκτυπώσουμε όλα τα παραπάνω μέσα σε έναν πίνακα χαρακτήρων
  - αντί για την έξοδο του προγράμματος

```
/* ασφαλής ανάγνωση string */

#include <stdio.h>
#define SIZE 32
#define FORMAT_SIZE 16

int main(int argc, char *argv[]) {
  char str[SIZE]; /* string που θα διαβαστεί */
  char fstr[FORMAT_SIZE];  /* format string για την scanf */

  snprintf(fstr, FORMAT_SIZE, "%%%ds", SIZE-1);
  printf("scan with format string %s\n",fstr);

  printf("enter string (up to %d chars): ",SIZE-1);
  scanf(fstr,str);

  printf("str is %s\n", str);

  return(0);
}
```

Ορίσματα προγράμματος

Παράμετροι συνάρτησης main

Ορίσματα προγράμματος

- Ένα πρόγραμμα μπορεί να δέχεται ως παραμέτρους εκκίνησης τα λεγόμενα **ορίσματα (arguments)**

- Τα ορίσματα του προγράμματος δίνονται από την γραμμή εντολών
  - οτιδήποτε ακολουθεί μετά το όνομα του προγράμματος

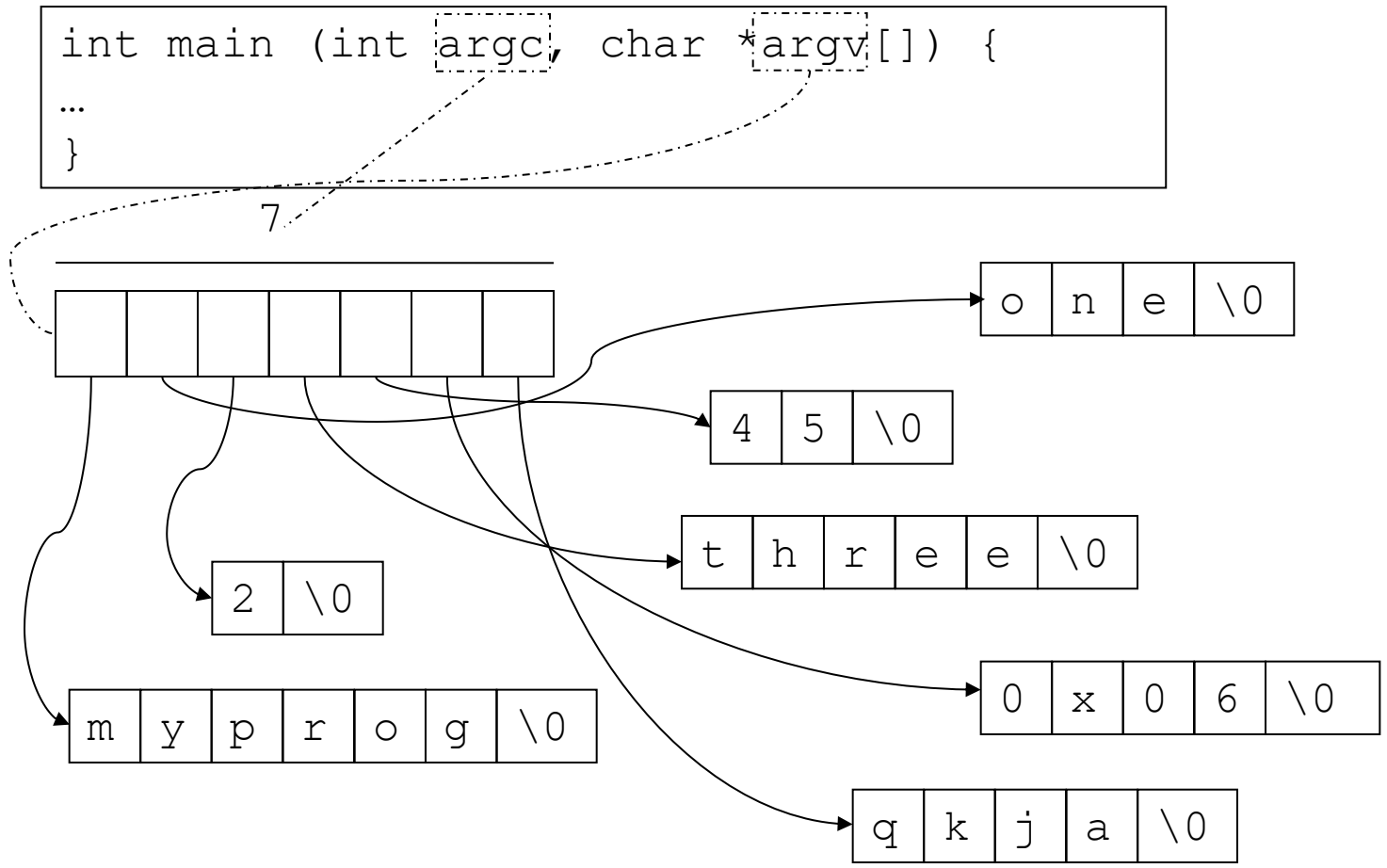
- Τα ορίσματα του προγράμματος **δεν** έχουν σχέση με την είσοδο του προγράμματος
  - χρησιμοποιούνται για να παραμετροποιηθεί η εκτέλεση του προγράμματος ή/και να επιλεγούν/ενεργοποιηθούν κάποιες συγκεκριμένες εναλλακτικές ή επιπρόσθετες λειτουργίες του

Παράμετροι συνάρτησης `main`

- Τα ορίσματα του προγράμματος περνιούνται στο πρόγραμμα (από το περιβάλλον εκτέλεσης)
- Ως μια σειρά από **συμβολοσειρές** (strings)
  - μέσω των **παραμέτρων** της `main`

1. **`int argc`** (argument count)
   - αριθμός ορισμάτων που περάστηκαν στο πρόγραμμα
   - συμπεριλαμβανομένου του ονόματος του (πρώτο όρισμα)
2. **`char *argv[]`** (argument vector)
   - πίνακας από δείκτες σε string (`array-of-pointer-to-char`)
   - το `argv[i]` περιέχει ένα **δείκτη** στη θέση μνήμης του `i`-οστού ορίσματος που δόθηκε από την γραμμή εντολών
   - το `argv[0]` είναι κατά σύμβαση το όνομα του προγράμματος

```
>./myprog one 2 three 45 0x06 qkja
```



Μετατροπή string σε `int` / `double`

- Κατάλληλες συναρτήσεις προσφέρονται από την βιβλιοθήκη «συνηθισμένων λειτουργιών» `stdlib`
- `#include<stdlib.h>`

- `int atoi(const char *nptr);`
- `long atol(const char *nptr);`
  - μετατρέπουν τους **πρώτους** χαρακτήρες του string όπου δείχνει η παράμετρος `nptr` σε μια τιμή τύπου `int` / `long int` αντίστοιχα

- `double atof(const char *nptr);`
  - μετατρέπει τους **πρώτους** χαρακτήρες του string όπου δείχνει η παράμετρος `nptr` σε μια τιμή `double`

```
#include <stdio.h>
#include <stdlib.h>

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

  for (i=1; i<argc; i++) {
    printf("%s\n", argv[i]);
    printf("%d\n", atoi(argv[i]));
    printf("%lf\n", atof(argv[i]));
  }

  return(0);
}
```

```c
#include <stdio.h>
#include <string.h> /* To use string functions */
#include <stdlib.h>

#define NOFARGS 4

int main(int argc, char *argv[]) {
  int res, num1, num2;

  num1 = atoi(argv[1]);   /* Convert string to int */
  num2 = atoi(argv[3]);

  if (!strcmp(argv[2],"+"))/*Like an == for strings*/
    res = num1 + num2;
  else if (!strcmp(argv[2],"-"))
    res = num1 - num2;
  else {
    printf("<op> must be + or -\n");
    return(1);
  }

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