

Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

Συναρτήσεις βασικής βιβλιοθήκης για αλφαριθμητικά

- πρότυπα στο `string.h`
- `char *strcpy (char *, const char *) ;`
- `int strcmp (const char *, const char *) ;`
- `char *strcat (char *, const char *) ;`
- `char * strtok(char *, const char *) ;`
- `char *strchr (const char *, char) ;`
- `size_t strcspn(const char *, const char *) ;`
- `size_t strlen (const char *) ;`
- και άλλες...

`char * strpbrk (const char *, const char *);`

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

```
int main(void) {
    char str[ ] = "testabcabc";
    char chars[] = "ac";
    char * str_i;

    str_i = strpbrk(str, chars);
    while (str_i!=NULL) {
        printf("%2d %c %p %d\n", str_i-str, *str_i, str_i, strlen(str_i));
        str_i = strpbrk(str_i+1, chars);
    }

    return 0;
}
```

`char * strpbrk (const char *, const char *);`

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

```
int main(void) {
    char str[ ] = "testabcabc";
    char chars[] = "ac";
    char * str_i;

    str_i = str;
    while ((str_i = strpbrk(str_i, chars))!=NULL) {
        printf("%2d %c %p %d\n", str_i-str, *str_i, str_i, strlen(str_i));
        str_i ++ ;
    }

    return 0;
}
```

`char * strstr(const char *, const char *);`

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

int main(void) {
    char str[ ] = "testabcabc";
    char * ch_ptr ;

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

    ch_ptr = strstr(str, "abc");
    strncpy(ch_ptr, "FGH", 3);

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

    return 0;
}
```

Buffer overflow!

```
#include <stdio.h>
#include <string.h>
#define N 64

int main(void) {

    char str2[] = "abcdefghijklmaaaaaaaaaa" ;
    char str1[10] = "copy!";
    char word2[] = "aaa";

    printf("%s %s %p %p\n", str2, str1, str2, str1);

    strcpy(str1, str2);

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

    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#define N 64

int main(void) {

    char str2[] = "abcdefghijklmaaaaaaaaa" ;
    char str1[10] = "copy!";
    char word2[] = "aaa";

    printf("%s %s %p %p\n", str2, str1, str2, str1);

    strncpy(str1, str2, sizeof str1);
    str1[9] = '\0';
    printf("%s %s\n", str2, str1);

    return 0;
}
```

```
char * strncpy(char *, const char *, size_t);
```

```
#include <stdio.h>
#include <string.h>
#define N 64
```

```
int main(void) {

    char str[N] ;
    char word[] = "copy!";
    char word2[] = "aaa";

    strcpy(str, word);
    printf("%s\n", str);

    strncpy(str+1, word, 2);

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

    strncpy(str, word2, 4);
    printf("%s\n", str);

    return 0;
}
```


Πίνακας αλφαριθμητικών

```
#define N 3
```

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

Ο πίνακας 2-D λειτουργεί ως
πίνακας 1-D με στοιχεία πίνακες 1-D.

```
int main ( ) {
```

```
    int i ;
```

```
    char text[N][11] ={"dokimi", "test", "paradeigma"};
```

```
    printf("text requires %d bytes\n", sizeof text ) ;
```

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

```
    return 0;
```

```
}
```

text[0]	'd'	'o'	'k'	'i'	'm'	'i'	0				
text[1]	't'	'e'	's'	't'	0						
text[2]	'p'	'a'	'r'	'a'	'd'	'e'	'i'	'g'	'm'	'a'	0

Πίνακας αλφαριθμητικών

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

```
#include <string.h>
```

```
int main (void) {  
    char words[5][9+1];  
    int i, j;  
  
    for (i=0; i<5; i++) {  
        printf("word %i:", i);  
        scanf("%9s", words[i]);  
    }  
  
    printf("\n");  
    for (i=0; i<5; i++) {  
        printf("%s ", words[i]);  
    }  
  
    return 0;  
}
```

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

int main () {
    char words[5][9+1];
    int i, j;

    for (i=0; i<5; i++) {
        printf("word %i:", i);
        scanf("%9s", words[i]);
    }
    printf("\n");
    for (i=0; i<5; i++) {
        printf("%s ", words[i]);
    }
    printf("\n");

    memmove( words[2], words[0], 2*sizeof words[0]);
    printf("\n");
    for (i=0; i<5; i++) {
        printf("%s ", words[i]);
    }

    return 0;
}
```

Παρεμβολή στη θέση 3

0 aa
1 bb
2 cc
3 dd
4 ee
5 ff
6 gg
7 hh

0 aa
1 bb
2 cc
3
4 dd
5 ee
6 ff
7 gg

0 aa
1 bb
2 cc
3 zz
4 dd
5 ee
6 ff
7 gg

```
#include <stdio.h>
#include <string.h>
#define N 4

int main () {
    int a[N] = {1, 2, 4, 8};
    int i;

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

    memmove(a+2, a + 1, 2*sizeof (int));
    for (i=0;i<N; i++) printf("%d ",a[i]);
    printf("\n");

    a[1] = 7;
    for (i=0;i<N; i++) printf("%d ",a[i]);
    printf("\n");

    return 0;
}
```

```
#define N 3
#include <stdio.h>
```

Πίνακας 1-D με στοιχεία δείκτες σε χαρακτήρα

```
int main ( ) {
    int i ;
    char text1[N][11]={"dokimi", "test", "paradeigma"};
    char *text2[N] = {"dokimi", "test", "paradeigma"};

    printf("text1 requires %d bytes\n", sizeof text1 ) ;
    printf("text2 requires %d bytes\n", sizeof text2 ) ;

    for (i=0 ; i < N ; printf ("%s ", text2[i++]));

    return 0;
}
```

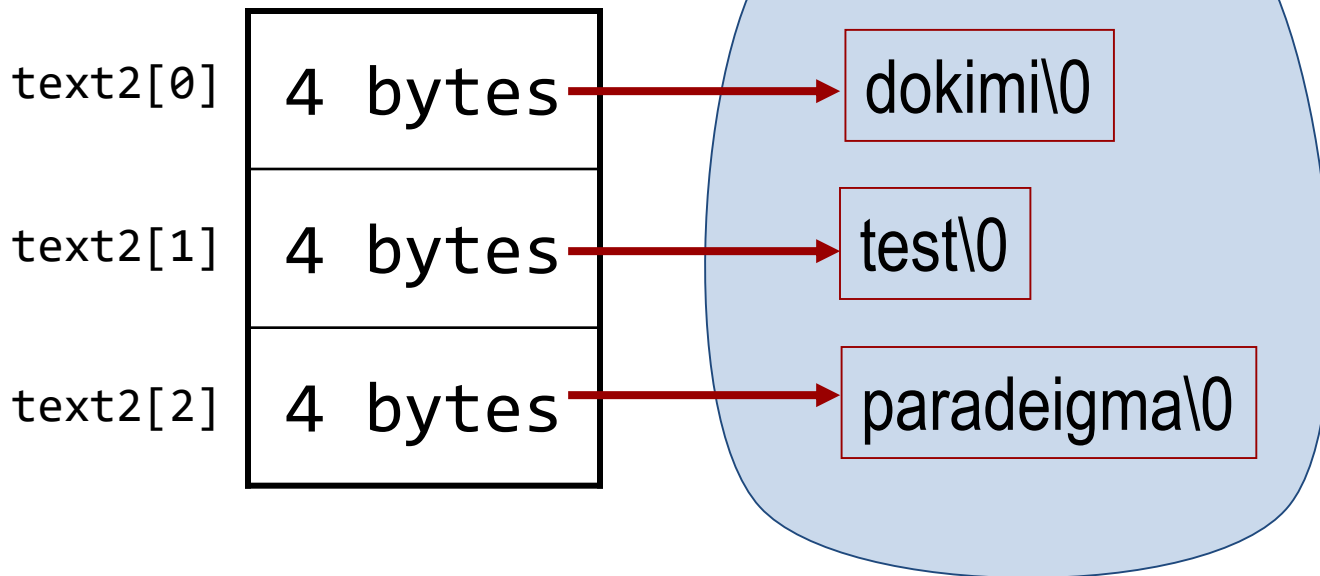
Προσοχή:
Δεν περιλαμβάνει
τις αλφαριθμητικές
σταθερές!

```
$ ./a
text1 requires 33 bytes
text2 requires 12 bytes
dokimi test paradeigma
Paliuras@MOB ~/nlect9
$
```

Οργάνωση μνήμης και πίνακες δεικτών σε χαρακτήρες

Οι αλφαριθμητικές σταθερές της αρχικοποίησης αποθηκεύονται σε άλλη περιοχή μνήμης.

```
char *text2[3];
```



text2: περιλαμβάνει τρεις διευθύνσεις, όχι τα ίδια τα αλφαριθμητικά.

Δήλωση και αρχικοποίηση

```
main ( ) {  
    char name[10] = "katerina";  
  
    printf ("%s", name);  
  
    scanf ("%s", name);  
  
    printf ("%s", name);  
  
}
```


Δήλωση και ανάθεση τιμής σε αλφαριθμητικό

```
char *strcpy(char *, const char*);
```

```
main ( ) {  
    char name[10];  
    name = "katerina";  
    printf ("%s", name);  
  
    scanf ("%s", name);  
    printf ("%s", name);  
}
```

Λάθος
(στη C)

```
#include <string.h>  
main ( ) {  
    char name[10];  
    strcpy(name, "katerina");  
    printf ("%s", name);  
  
    scanf ("%s", name);  
    printf ("%s", name);  
}
```

Σωστός τρόπος:

Τι γίνεται με δείκτες;

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

main ( ) {
char *name = "katerina";

printf ("%s", name);

}
```

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

main ( ) {
char *name ;

name = "katerina";

printf ("%s", name);

}
```

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

main ( ) {
char *name ;

name = "katerina";

printf ("%s", name);

*name = 'K';

printf ("%s", name);

}
```

**σφάλμα χρόνου εκτέλεσης (run-time error):
segmentation fault**

Λύση: χρήση διαθέσιμης περιοχής μνήμης (πχ με `calloc ()`)

Διεύθυνση ονόματος πίνακα

- `char word[5];`
- `word` είναι η διεύθυνση του πρώτου στοιχείου `&word[0] => char *`
- `&word`
 - έχει αριθμητικά την ίδια τιμή με το `word`
 - Είναι όμως τύπου `=> char (*)[5]`

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

```
int main( ) {
```

```
int array[5] = { 0, 1, 2, 3, 4} ;
```

```
printf("Have the same value...\n");
```

```
printf("array : %X\n", array);
```

```
printf("&array: %X\n", &array);
```

```
printf("...but not the same type:\n");
```

```
printf("next element (array+1): %X\n", array + 1);
```

```
printf("next element (&array+1): %X\n", &array + 1 );
```

```
return 0;
```

```
}
```

array, &array

Έχουν *ίδια* τιμή,

Αλλά *διαφορετικό* τύπο

- **int ***

- **int (*) [5]**

ίδια τιμή

Μια θέση ακεραίου μετά

20 θέσεις ακεραίου μετά
Δηλ. (1 πίνακας array)

```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
Have the same value...
array : 22FE40
&array: 22FE40
...but not the same type:
next element (array+1): 22FE44
next element (&array+1): 22FE54
```

- `char manywords[3][5];`
- Ένας πίνακας 3 στοιχείων,
 - καθένα από τα οποία είναι πίνακας 5 στοιχείων,
 - καθένα από τα οποία είναι `char`.

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

```
int main( ) {
```

```
    int i;
```

```
    char manywords[3][5];
```

← Πίνακας δύο διαστάσεων

```
    printf("array starts at memory address %X\n", manywords);
```

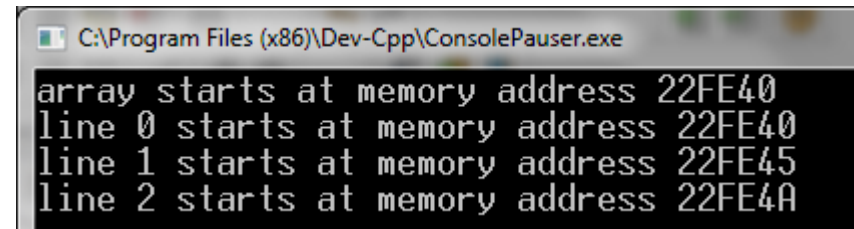
```
    for (i = 0; i < 3; i++)
```

```
        printf("line %d starts at memory address %X\n",  
              i, manywords[i]);
```

```
    return 0;
```

```
}
```

← Διεύθυνση αρχής της i-οστής γραμμής



```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe  
array starts at memory address 22FE40  
line 0 starts at memory address 22FE40  
line 1 starts at memory address 22FE45  
line 2 starts at memory address 22FE4A
```

Πίνακες πολλών διαστάσεων ως παράμετροι σε συναρτήσεις

`char a[3][2];`

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

`char b[2][3];`

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

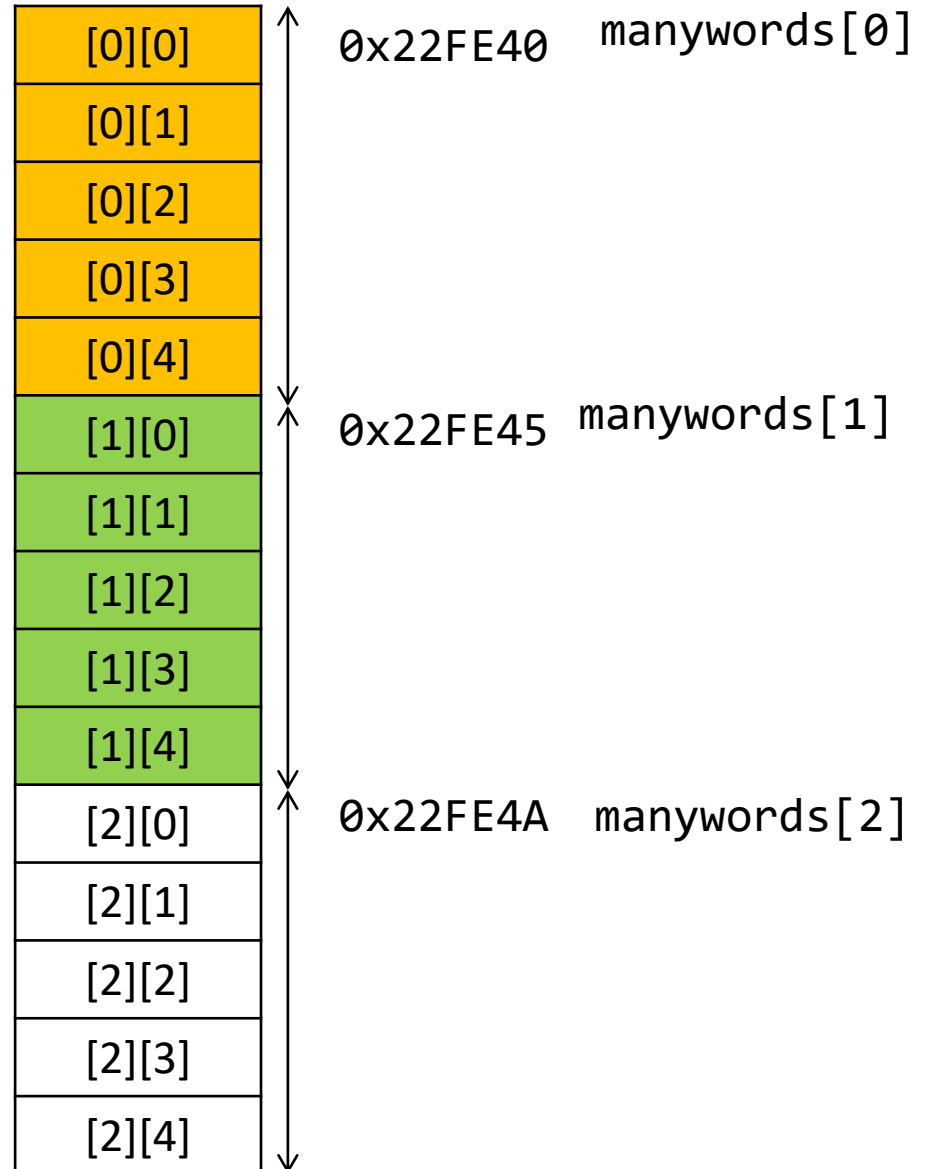
- Οι συναρτήσεις χειρίζονται τους πίνακες κατ' αναφορά
- Η θέση ενός στοιχείου στη μνήμη εξαρτάται από τη γεωμετρία του πίνακα
 - Το `a[1][0]` είναι το 3^ο στοιχείο, το `b[1][0]` είναι το 4^ο στοιχείο
- Μια συνάρτηση πρέπει να ξέρει τη γεωμετρία ενός πίνακα παραμέτρου
 - Μπορούμε να παραλείψουμε μόνο την πρώτη διάσταση (το πλήθος γραμμών) σε μια δήλωση
 - Σε πίνακες N διαστάσεων, μπορούμε να παραλείψουμε μόνο μία διάσταση και να δηλώσουμε μήκη για τις υπόλοιπες $N - 1$
- Ευέλικτος κώδικας χρησιμοποιώντας ως παράμετρο δείκτη στο πρώτο στοιχείο του πίνακα

```
char manywords[3][5];
```

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

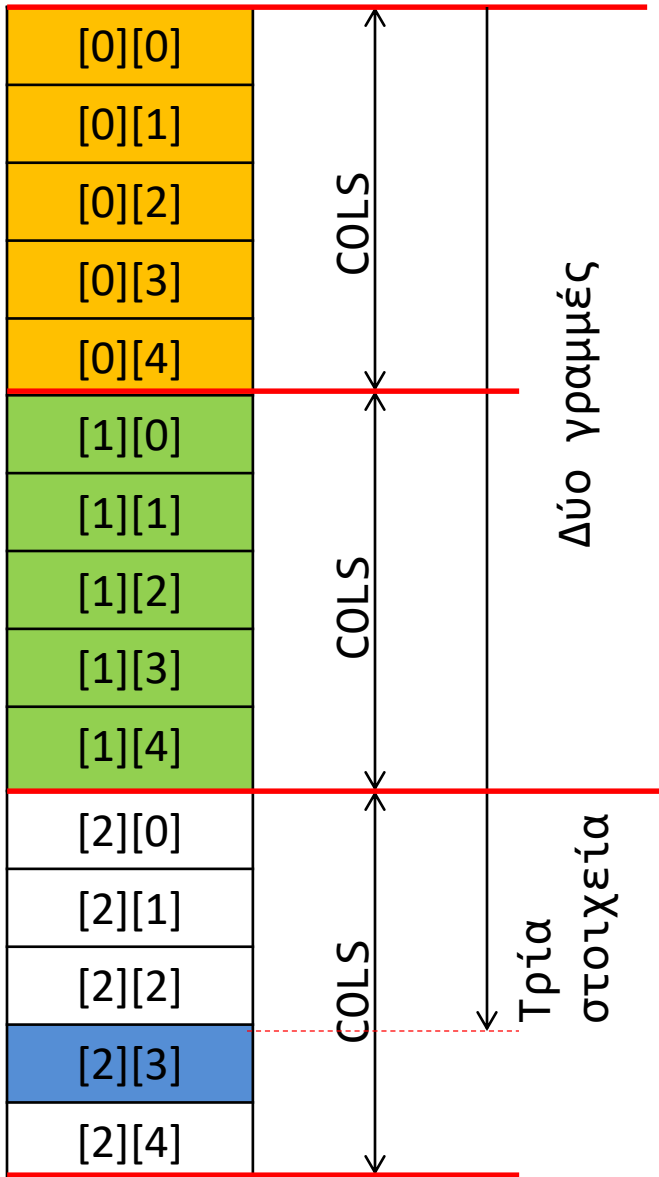
```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe  
array starts at memory address 22FE40  
line 0 starts at memory address 22FE40  
line 1 starts at memory address 22FE45  
line 2 starts at memory address 22FE4A
```

Αποθήκευση στη μνήμη



Το μέγεθος σε bytes ανά στοιχείο καθορίζεται από τον τύπο του στοιχείου. Εδώ **char**, άρα 1 byte ανά στοιχείο.

```
#define ROWS 3
#define COLS 5
char manywords[ROWS][COLS];
```



<code>[0][0]</code>	<code>[0][1]</code>	<code>[0][2]</code>	<code>[0][3]</code>	<code>[0][4]</code>
<code>[1][0]</code>	<code>[1][1]</code>	<code>[1][2]</code>	<code>[1][3]</code>	<code>[1][4]</code>
<code>[2][0]</code>	<code>[2][1]</code>	<code>[2][2]</code>	<code>[2][3]</code>	<code>[2][4]</code>

Κάθε γραμμή έχει `COLS` στοιχεία

Το στοιχείο `manywords[2][3]` βρίσκεται δύο γραμμές (άρα $2 * 5$ στοιχεία) συν τρία στοιχεία της τρέχουσας γραμμής από την αρχή του πίνακα

Το στοιχείο `manywords[i][j]` βρίσκεται στη θέση
`(char *)manywords + i*COLS+j`

- `int numbers[2][3][4];`
- Ένας πίνακας 2 στοιχείων,
 - καθένα από τα οποία είναι πίνακας 3 στοιχείων,
 - καθένα από τα οποία είναι πίνακας 4 στοιχείων
 - καθένα από τα οποία είναι `int`

[0][0][0]
[0][0][1]
[0][0][2]
[0][0][3]
[0][1][0]
[0][1][1]
[0][1][2]
[0][1][3]
[0][2][0]
[0][2][1]
[0][2][2]
[0][2][3]
[1][0][0]
[1][0][1]
[1][0][2]
[1][0][3]
[1][1][0]
[1][1][1]
[1][1][2]
[1][1][3]
[1][2][0]
[1][2][1]
[1][2][2]
[1][2][3]

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

```
void readarray(int *, int, int, int);
void printarray(int *, int, int, int);
void print3D(int [2][3][4]);
```

```
int main( ) {
```

```
    int array3D[2][3][4];
```

```
    readarray( (int *)array3D, 2, 3, 4);
    printarray((int *)array3D, 2, 3, 4);
    print3D(array3D);
```

```
    return 0;
```

```
}
```

Εδώ η γεωμετρία του πίνακα είναι παράμετρος

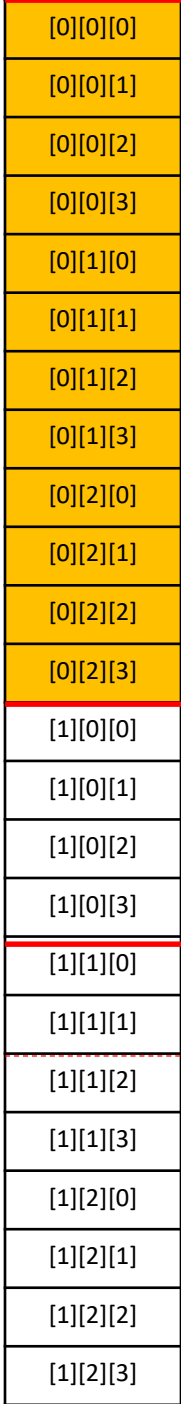
Χειρίζεται πίνακες τύπου `int [2][3][4]` μόνο

Πίνακας σταθερής γεωμετρίας ως παράμετρος

```
void print3D(int data[2][3][4]) {
    int i, j, k;

    for (i = 0; i<2; i++) {
        for (j=0; j<3; j++) {
            for (k=0; k<4; k++)
                printf("%2d ", data[i][j][k]);
            printf("\n");
        }
        printf("\n");
    }
    return 0;
}
```

Χειρίζεται πίνακες τύπου `int [2][3][4]` μόνο



```
int array3D[2][3][4];
           h w d

(int *) array3D + i*w*d + j * d + k
```

$$1*w*d = 1 * 3 * 4$$

$$1*d = 1 * 4$$

2

```
Το στοιχείο array3D[1][1][2]
βρίσκεται στη θέση
(int *) array3D + 1*3*4+ 1*4 + 2
```

Πίνακας σταθερού πλήθους διαστάσεων, όχι προκαθορισμένης γεωμετρίας, ως παράμετρος σε συνάρτηση

```
void readarray(int *data, int h, int w, int d) {
int i, j, k ,count = 0 ;
    for (i = 0; i<h; i++)
        for (j=0; j<w; j++)
            for (k=0; k<d; k++)
                *(data + i*w*d + j * d + k) = count ++;
}

void printarray(int *data, int h, int w, int d) {
int i, j, k ;
    for (i = 0; i<h; i++) {
        for (j=0; j<w; j++) {
            for (k=0; k<d; k++)
                printf("%2d ", *(data + i*w*d + j*d + k));
            printf("\n");
        }
        printf("\n");
    }
}
```

Αναφερόμαστε στο στοιχείο [i][j][k] ενός πίνακα [h][w][d]

[0][0][0]

[0][0][1]

[0][0][2]

[0][0][3]

[0][1][0]

[0][1][1]

[0][1][2]

[0][1][3]

[0][2][0]

[0][2][1]

[0][2][2]

[0][2][3]

[1][0][0]

[1][0][1]

[1][0][2]

[1][0][3]

[1][1][0]

[1][1][1]

[1][1][2]

[1][1][3]

[1][2][0]

[1][2][1]

[1][2][2]

[1][2][3]

```
void myread(int *data, int h, int w, int d) {  
    int i=0, count = 0;  
    for (; i< h*w*d; *(data + (i++)) = count ++ );  
}
```

```
void myread(int *data, int h, int w, int d) {  
    int i=0;  
    for (; i< h*w*d; *(data + i) = i++ );  
}
```

```
void myread(int *data, int h, int w, int d) {  
    int i=0;  
    for (; i< h*w*d; *(data ++ ) = i++ );  
}
```

```
void myread(int *data, int h, int w, int d) {  
    int i=0;  
    for (; (*(data ++ ) = i++ )< h*w*d;);  
}
```

Σε *μερικές* περιπτώσεις μπορούμε να κάνουμε αντίστοιχα πράγματα
Με λιγότερο κώδικα, αξιοποιώντας τον τρόπο αποθήκευσης στη μνήμη.

Ευέλικτος κώδικας

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

```
void readarray(int *, int, int, int);
void printarray(int *, int, int, int);
void print3D(int [2][3][4]);
```

```
int main( ) {
```

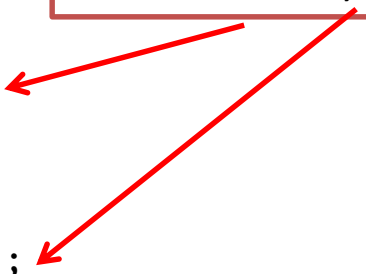
```
    int array3D[2][3][4];
    int another3D[2][2][2];
```

```
    readarray( (int *)array3D, 2, 3, 4);
    printarray((int *)array3D, 2, 3, 4);
    print3D(array3D);
```

```
    readarray( (int *)another3D, 2, 2, 2);
    printarray((int *)another3D, 2, 2, 2);
    return 0;
```

```
}
```

Ίδια συνάρτηση, η γεωμετρία ως
του πίνακα ως παράμετρος.



Πίνακες πολλών διαστάσεων

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

```
int main( )
```

```
{
```

```
    int array2d[][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

```
    int i, j;
```

```
    printf("Size of array in bytes: %d\n", sizeof array2d);
```

```
    printf("Size of an element: %d\n", sizeof array2d[0]);
```

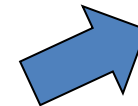
```
    for (i=0; i< 3; i++)
```

```
        for (j=0; j< 4; j++)
```

```
            printf("%d %d %d\n", i, j, array2d[i][j]);
```

```
    return 0;
```

```
}
```



48 bytes

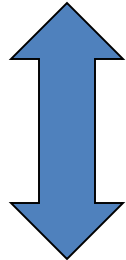


16 bytes

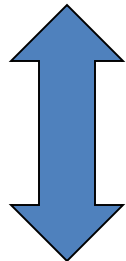

```
for (i=0; i < 3; i++)
    for (j=0; j < 4; j++)
        printf("%d %d %d\n", i, j, *( array2d[i] + j));
```

```
for (i=0; i < 3; i++)
    for (j=0; j < 4; j++)
        printf("%d %d %d\n", i, j, *( *(array2d + i) + j));
```

```
for (i=0; i < 3; i++)
    for (j=0; j < 4; j++)
        printf("%d %d %d\n", i, j, *((int *) array2d + i * 4 + j));
```



Το i μετράει γραμμές



4 ακέραιοι ανά γραμμή

**Αναλυτική αλλαγή τύπου σε (int *)
⇒ Γίνεται διεύθυνση ακεραίου**

Το C99 επιτρέπει Variable Length Arrays (VLAs)

```
void test(int rows, int cols, int x[rows][cols]) {  
    /* ... */  
}
```

```

#include <stdio.h>
#include <stdlib.h>
//
// C99 comment style
// Demonstration of C99 VLAs
//
void readarray(int rows, int cols, int x[rows][cols]);
void printarray(int rows, int cols, int x[rows][cols]);

int main(int argc, char *argv[]) {
    int a[5][5];
    readarray(5, 5, a);
    printarray(3, 5, a);
    return 0;
}

void readarray(int rows, int cols, int x[rows][cols]) {
    int i, j;

    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            x[i][j] = -i*cols - j ;
}

void printarray(int rows, int cols, int x[rows][cols]) {
    int i, j;

    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++)
            printf("%4d", x[i][j]) ;
        printf("\n");
    }
}

```

Μηχανισμοί κλήσης συναρτήσεων

Τι θα τυπωθεί;

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

```
int myfun(int, int);
```

```
main ( ) {
```

```
int a = 3, b = 3;
```

```
myfun(a, b);
```

```
printf("main: a:%d b:%d\n", a,  
      b);
```

```
}
```

```
int myfun(int a, int b) {
```

```
a++;
```

```
b++;
```

```
printf("myfun: a:%d b:%d\n",  
      a, b);
```

```
return 0;
```

```
}
```

Κλήση συνάρτησης κατ' αναφορά (call by reference)

- πρότυπο: **void swar(int *a, int *b);**
- κλήση: **swar(&value1, &value2);**

1. Η συνάρτηση επενεργεί **απευθείας στις θέσεις μνήμης** των μεταβλητών που χρησιμοποιούνται ως πραγματικά ορίσματα.
2. **Δεν** δημιουργούνται τοπικά αντίγραφα των δεδομένων.
 - Ισχύει για τις διευθύνσεις;
3. Στη C, ουσιαστικά, υλοποιείται ως κατ' αξία πέρασμα διευθύνσεων.

Ορισμός συνάρτησης swap()

```
void swap(int *a, int *b) {  
    int temp ;  
    temp = *b ;  
    *b = *a ;  
    *a = temp ;  
}
```

Παράδειγμα χρήσης κλήσης κατ' αναφορά

```
#include <stdio.h>
void swap (int *, int *);


main ( ) {
    int value1 = 5;
    int value2 = 3;

    printf("value1: %d value2: %d\n", value1, value2);

    swap (&value1, &value2);

    printf("value1: %d value2: %d\n", value1, value2);
}
```

```
void swap(int *a , int *b) {
    int temp;
    temp = *a ;
    *a = *b ;
    *b = temp ;
}
```



```
Paliuras@MOB ~/nlect9
$ ./a
value1: 5 value2: 3
value1: 3 value2: 5
```


Μερικές διαφορές

- Μηχανισμός κλήσης κατ' αξία (call by value)
 - δημιουργούνται τοπικά αντίγραφα των τιμών των ορισμάτων
 - Αν τα ορίσματα έχουν μήκος πολλών bytes, επιβαρύνεται η εκτέλεση.
 - Η συνάρτηση δεν μπορεί να αλλάξει τις τιμές ορισμάτων στο σημείο κλήσης.
- Μηχανισμός κλήσης με αναφορά (call by reference)
 - δεν δημιουργούνται τοπικά αντίγραφα τιμών,
 - μόνο των διευθύνσεων !!!
 - η συνάρτηση ενημερώνεται για τη θέση μνήμης στην οποία είναι αποθηκευμένο ένα όρισμα.
 - μπορεί να είναι ταχύτερο
 - είναι δυνατόν να αλλάξουν οι τιμές ορισμάτων στο σημείο κλήσης.
 - χρειάζεται προσοχή, μπορεί να γίνει εκ παραδρομής.
 - Είναι ένας τρόπος να επιστρέψω περισσότερες από μία τιμές
 - Για τη C, καταχρηστικά λέγεται έτσι : επί της ουσίας είναι κατ' αξία πέρασμα δεικτών.

```
#include <stdio.h>
#include <stdlib.h>
void sumdiff(int, int, int *, int *);
```

Δείκτες σε ακέραιο

```
int main(int argc, char *argv[])
{
int a = 3, b =4 ;
int sum;
int diff;
```

Διευθύνσεις των μεταβλητών που
θα αλλάξουν τιμές

```
sumdiff(a,b, &sum, &diff);

printf("%d %d\n", sum, diff);

system("PAUSE");
return 0;
}
```

Η s αρχικοποιείται στην τιμή &sum

```
void sumdiff(int a, int b, int *s, int *d) {
*s = a + b;
*d = a - b;
a++;
b++;
printf("%d %d\n",a,b);
}
```

Αλλάζουν τιμές τα **περιεχόμενα** των
θέσεων με **διευθύνσεις** s, d.

Στη sumdiff()
ορίζονται **νέες**
μεταβλητές a, b

Οι a, b της
main() είναι
διαφορετικές
μεταβλητές

Εμβέλεια ονομάτων

```
int i, k ;
```

```
main () { int i, m; }
```

```
int f1 () {int i, j;}
```

```
int f2 () {int n;}
```

```
#include <stdio.h>
#include <stdlib.h>
int f1(int);
```

καθολική

```
int i = 5;
```

Τι θα γίνει αν αφαιρεθεί
το /* */ από τη δήλωση;

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

```
    /*int i = 2;*/
    int k = 1;
    i++;
```

τοπική

```
    f1(i);
    f1(k);
    system("PAUSE");
    return 0;
```

```
}
```

καθολική i

```
int f1(int m) {
    printf("%d\n", m+i);
    return 0;
```

Τοπική m

```
}
```

Διάρκεια ζωής μεταβλητής

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

void function (void);
void function1 (void);
int main () {

    function1();
    function ( ) ;
    function1 ( ) ;
    function ( ) ;

    return 0;
}

void function1() {
int j = 0;
    j++;
    printf("function1, j: %d at %X\n", j, &j);
}

void function ( ) {
int i ;
    i++ ;
    printf("function, i: %d at %X\n", i, &i);
    printf("%d\n", i);
}
```

Η τοπική μεταβλητή
i δεν αρχικοποιείται!

Τι βγαίνει εδώ;

i, *j*, είναι τοπικές μεταβλητές

Διάρκεια ζωής: όσο εκτελείται
η συνάρτηση

Η θέση μνήμης επαναχρησιμοποιείται

Διάρκεια μεταβλητής (2)

```
void function1 (void);
void function2 (void);
void function(void);
main ( ) {
    function();
    function1( ) ;
    function2( ) ;
    function1( ) ;
    function2( ) ;
    function1( ) ;
    function2( ) ;
}
void function() {
int k = 0;
}
void function1( ) {
int i;
    i++;
    printf("f1:%d\n", i);
}
void function2( ) {
int j;
    j++;
    printf("f2:%d\n", j);
}
```

Αν η κλήση της function1() γίνεται
εναλλάξ με την function2(), η
συμπεριφορά αλλάζει...

Εμφανίζονται εξαρτήσεις
(είναι δυνατόν να ...)

```

void function1 (void);
void function2 (void);

main ( ) {
    function1( ) ;
    function2( ) ;
    function1( ) ;
    function2( ) ;
    function1( ) ;
    function2( ) ;
}
void function1 ( ) {
int i = 0;
    i++;
    printf("f1:%d\n", i);
}
void function2 ( ) {
int j = 0;
    j++;
    printf("f2:%d\n", j);
}

```

Λύση;;;

Αρχικοποιώντας τις μεταβλητές
κάθε φορά που καλείται η συνάρτηση,
οι τοπικές μεταβλητές μηδενίζονται κάθε
φορά που καλείται η συνάρτηση.

```

void function1 (void);
void function2 (void);

main ( ) {
    function1( ) ;
    function2( ) ;
    function1( ) ;
    function2( ) ;
    function1( ) ;
    function2( ) ;
}
void function1 ( ) {
static int i = 0;
    i++;
    printf("f1:%d\n", i);
}
void function2 ( ) {
static int j = 0;
    j++;
    printf("f2:%d\n", j);
}

```

Η λέξη κλειδί **static**
σε δηλώσεις τοπικών
μεταβλητών

static: ο χώρος μνήμης της
μεταβλητής δεν αποδεσμεύεται
όταν ολοκληρωθεί μια εκτέλεση
της συνάρτησης.

Άλλη χρήση της **static**: Ορισμός εμφάνισης αρχείου

- αρχείο πηγαίου κώδικα: code1.c

```
int func (int);  
void report(void);
```

```
main () {
```

```
int i ;
```

```
for ( i=0; i< 5; i++)
```

```
    printf("%d\n", func(i));
```

```
report();
```

```
}
```

- αρχείο πηγαίου κώδικα: code2.c

```
static int sum = 0;
```

```
int func(int k) {
```

```
int i;
```

```
for (i=0;i<5; i++)
```

```
    sum += k ; /* sum = sum + k */
```

```
return 2 * k;
```

```
}
```

```
void report() {
```

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

```
}
```