

fprintf(): «Τύπωσε στο αρχείο»

- formatted **print** output, to **file**
- Ακριβώς οι ίδιοι μορφολογικοί κανόνες με εκείνους της *printf()*

```
float flot;  
int cnt, k;  
FILE *pF;      /* δήλωση ενός pointer-to-FILE */  
char msg[40]="This is my song!\n"; /* string */
```

```
pF = fopen("testfile.txt","w");  
cnt = fprintf(pF,"%s,yep!%d,%f,\n",msg,21,34.5);  
fclose(pF);
```

Άνοιγμα αρχείου για εγγραφή

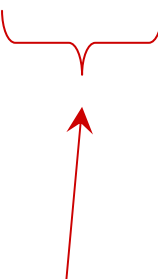
fprintf(): «Τύπωσε στο αρχείο»

```
int cnt;  
FILE *pF;    /* δήλωση ενός pointer-to-FILE */  
char msg[40]="This is my song!\n"; /* string */  
  
pF = fopen("testfile.txt","w");  
cnt = fprintf(pF,"%s,yep!%d,%f,\n",msg,21,34.5);  
fclose(pF);
```

Εγγραφή string, όπως ακριβώς στην *printf*

fprintf(): «Τύπωσε στο αρχείο»

```
int cnt;  
FILE *pF;    /* δήλωση ενός pointer-to-FILE */  
char msg[40]="This is my song!\n"; /* string */  
  
pF = fopen("testfile.txt","w");  
cnt = fprintf(pF,"%s,yep!%d,%f,\n",msg,21,34.5);  
fclose(pF);
```



Λίστα από pointers-to-items προς εγγραφή: string, int, float,...

fprintf(): «Τύπωσε στο αρχείο»

```
int cnt;  
FILE *pF;    /* δήλωση ενός pointer-to-FILE */  
char msg[40]="This is my song!\n"; /* string */  
  
pF = fopen("testfile.txt","w");  
cnt = fprintf(pF,"%s,yep!%d,%f,\n",msg,21,34.5);  
fclose(pF);
```

**Επιστρεφόμενη τιμή:
Ο αριθμός των bytes που
ενεγράφησαν στο αρχείο**

fprintf(): «Τύπωσε στο αρχείο»

```
int cnt;  
FILE *pF;    /* declare a pointer-to-FILE */  
char msg[40]="This is my song!\n"; /* string */
```

```
pF = fopen("testfile.txt","w");  
cnt = fprintf(pF,"%s,yep!%d,%f,\n",msg,21,34.5);  
fclose(pF);
```

**Τέλος εργασιών, κλείσιμο
του αρχείου**

fscanf(): «Διάβασε κείμενο από αρχείο»

- formatted **scan**ned input, from **f**ile
- Ακριβώς οι ίδιοι μορφολογικοί κανόνες με εκείνους της **scanf()**

```
float flot;  
int cnt, k;  
FILE *pF; /* δήλωση ενός pointer-to-FILE */  
char msg[40];  
pF = fopen("testfile.txt", "r"); assert(pF!=NULL);  
cnt = fscanf(pF, "%s %d %f", msg, &k, &flot);  
fclose(pF);
```

Άνοιγμα αρχείου για ανάγνωση

fscanf(): «Διάβασε κείμενο από αρχείο»

```
float flot;  
int cnt, k;  
FILE *pF; /* δήλωση ενός pointer-to-FILE */  
char msg[40];  
  
pF = fopen("testfile.txt","r"); assert(pF!=NULL);  
cnt = fscanf(pF,"%s %d %f",msg,&k,&flot);  
fclose(pF);
```

pointer-to-FILE



fscanf(): «Διάβασε κείμενο από αρχείο»

```
float flot;  
int cnt, k;  
FILE *pF;    /* δήλωση ενός pointer-to-FILE */  
char msg[40];
```

```
pF = fopen("testfile.txt","r");assert(pF!=NULL);  
cnt = fscanf(pF,"%s %d %f",msg,&k,&flot);  
fclose(pF);
```

Ανάγνωση string, όπως ακριβώς στην scanf, χωρίς &

fscanf(): «Διάβασε κείμενο από αρχείο»

```
float flot;  
int cnt, k;  
FILE *pF; /* δήλωση ενός pointer-to-FILE */  
char msg[40];
```

```
pF = fopen("testfile.txt","r");assert(pF!=NULL);  
cnt = fscanf(pF,"%s %d %f",msg,&k,&flot);  
fclose(pF);
```

**Λίστα από pointers-to-items προς ανάγνωση:
int, float,...**

fscanf(): «Διάβασε κείμενο από αρχείο»

```
float flot;  
int cnt, k;  
FILE *pF;    /* δήλωση ενός pointer-to-FILE */  
char msg[40];
```

```
pF = fopen("testfile.txt","r");assert(pF!=NULL);  
cnt = fscanf(pF,"%s %d %f",msg,&k,&flot);  
fclose(pF);
```

Επιστρεφόμενη τιμή: Ο αριθμός των στοιχείων που ανεγνώσθησαν. Σε περίπτωση σφάλματος θα επιστραφεί το μηδέν ή το EOF

fscanf(): «Διάβασε κείμενο από αρχείο»

```
float flot;  
int cnt, k;  
FILE *pF; /* δήλωση ενός pointer-to-FILE */  
char msg[40];
```

```
pF = fopen("testfile.txt","r");  
cnt = fscanf(pF,"%s %d %f",msg,&k,&flot);  
fclose(pF);
```

**Τέλος εργασιών, κλείσιμο του
αρχείου**

putc(): «Τύπωσε χαρακτήρα στο αρχείο»

- Πρωτότυπο της συνάρτησης:

```
int putc(int ch, FILE *pF);
```

όπου **pF** ο δείκτης αρχείου που επιστρέφεται από την **fopen** και **ch** είναι ο προς εγγραφή χαρακτήρας. Για ιστορικούς λόγους ο **ch** ονομάζεται **int** αλλά χρησιμοποιεί μόνο ένα byte, το byte χαμηλής τάξης (**ανοίξτε το `stdio.h` για να το επιβεβαιώσετε**).

- Εάν η λειτουργία της συνάρτησης επιτύχει, επιστρέφεται ο χαρακτήρας που γράφτηκε. Αν αποτύχει, θα επιστρέψει το **EOF** (**End Of File**, **τέλος αρχείου**, ακέραιος με τιμή -1).

getc(): «Διάβασε χαρακτήρα από αρχείο»

- Είναι συμπληρωματική της **putc()**. Πρωτότυπο της συνάρτησης:

```
int getc(FILE *pF);
```

όπου **pF** ο δείκτης αρχείου που επιστρέφεται από την **fopen**. Για ιστορικούς λόγους η **getc()** επιστρέφει έναν ακέραιο αλλά τα bytes υψηλής τάξης είναι μηδέν, άρα μόνο το byte χαμηλής τάξης περιέχει πληροφορία.

- Η συνάρτηση επιστρέφει ένα **EOF** όταν ο υπολογιστής φθάσει στο τέλος του αρχείου. Έτσι, για να διαβάσουμε ένα αρχείο κειμένου έως το σημάδι τέλους αρχείου, μπορούμε να χρησιμοποιήσουμε τον ακόλουθο κώδικα:

```
ch = getc(pF);
```

```
while (ch!=EOF){ ch=getc(pF);}
```

Ανάγνωση/εγγραφή ενός χαρακτήρα

```
FILE *pFin,*pFout;  
char k;
```

```
pFin = fopen("src.txt","r");  
pFout = fopen("dest.txt","w");  
if (pFin == NULL) return(-1);  
k = getc(pFin); /* διάβασε τον πρώτο χαρακτήρα ως int */  
while(k!=EOF)  
{  
    putc(k, pFout);  
    k = getc(pFin);  
}  
fclose(pFin); fclose(pFout);
```

Ανοίγει ένα αρχείο για
ανάγνωση (**r**ead) κι ένα για
εγγραφή (**w**rite)

Ανάγνωση/εγγραφή ενός χαρακτήρα

```
FILE *pFin,*pFout;  
char k;
```

```
pFin = fopen("src.txt","r");  
pFout = fopen("dest.txt","w");  
if (pFin == NULL) return(-1);  
k = getc(pFin); /* διάβασε τον πρώτο χαρακτήρα ως int */  
while(k!=EOF)  
{  
    putc(k, pFout);  
    k = getc(pFin);  
}  
fclose(pFin); fclose(pFout);
```

Διαβάζει τον πρώτο χαρακτήρα.
Η **getc()** επιστρέφει *int* ώστε να
μπορεί να γίνουν έλεγχος για
EOF (EOF=-1)

Ανάγνωση/εγγραφή ενός χαρακτήρα

```
FILE *pFin,*pFout;  
char k;
```

```
pFin = fopen("src.txt","r");  
pFout = fopen("dest.txt","w");  
if (pFin == NULL) return(-1);
```

```
k = getc(pFin); /* διάβασε τον πρώτο χαρακτήρα ως int */  
while (k!=EOF)
```

```
{  
    putc(k, pFout);  
    k = getc(pFin);  
}
```

```
fclose(pFin); fclose(pFout);
```

Εάν ο **pFin** έχει ένα **char**,
ο οποίος **ΔΕΝ ΕΙΝΑΙ**
EOF, τότε ...

Ανάγνωση/εγγραφή ενός χαρακτήρα

```
FILE *pFin,*pFout;  
char k;  
  
pFin = fopen("src.txt","r");  
pFout = fopen("dest.txt","w");  
if (pFin == pFout) return(-1);  
k = getc(pFin); /* διάβασε τον πρώτο χαρακτήρα ως int */  
while (k!=EOF)  
{  
    putc(k, pFout);  
    k = getc(pFin);  
}  
fclose(pFin); fclose(pFout);
```

← ... Τότε να αντιγραφεί στο **pFout**
και να ληφθεί ο επόμενος **char**, ...

Ανάγνωση/εγγραφή ενός χαρακτήρα

```
FILE *pFin,*pFout;  
char k;
```

```
pFin = fopen("src.txt","r");  
pFout = fopen("dest.txt","w");  
if (pFin == NULL) return(-1);  
k = getc(pFin); /* διάβασε τον πρώτο χαρακτήρα ως int */  
while(k!=EOF)  
{  
    putc(k, pFout);  
    k = getc(pFin);  
}  
fclose(pFin); fclose(pFout);
```

Όταν τελικά βρεθεί **EOF**,
περατώνεται η διαδικασία:
κλείνουν τα αρχεία.

Παράδειγμα: Να καταστρωθεί πρόγραμμα , το οποίο διαβάζει χαρακτήρες από το πληκτρολόγιο και τους γράφει σε αρχείο, έως ότου πληκτρολογήσουμε το σύμβολο του δολαρίου (\$).

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

```
main()
```

```
{
```

```
    FILE *pF; char ch;
```

```
    pF=fopen( "putc.res","w" );
```

```
    do
```

```
    {
```

```
        ch=getchar();
```

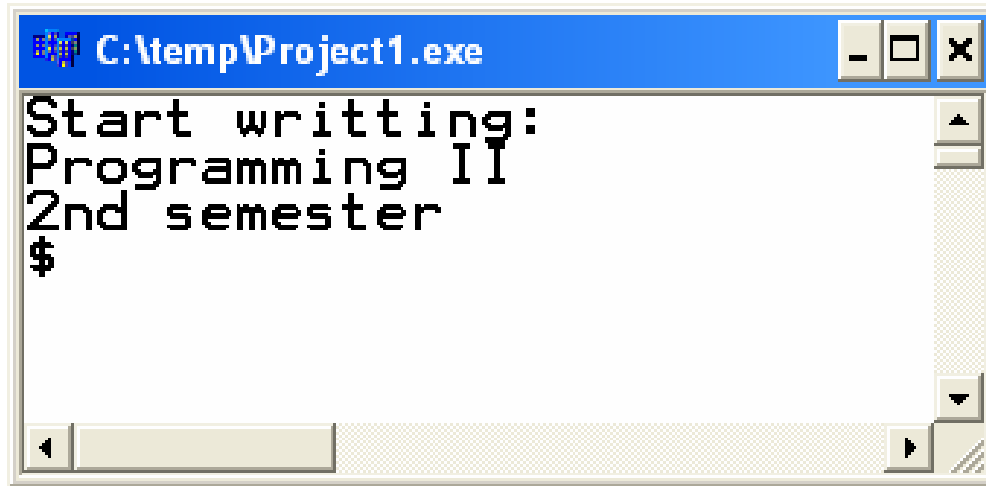
```
        putc(ch,pF);
```

```
    } while (ch!='$'); //end of do
```

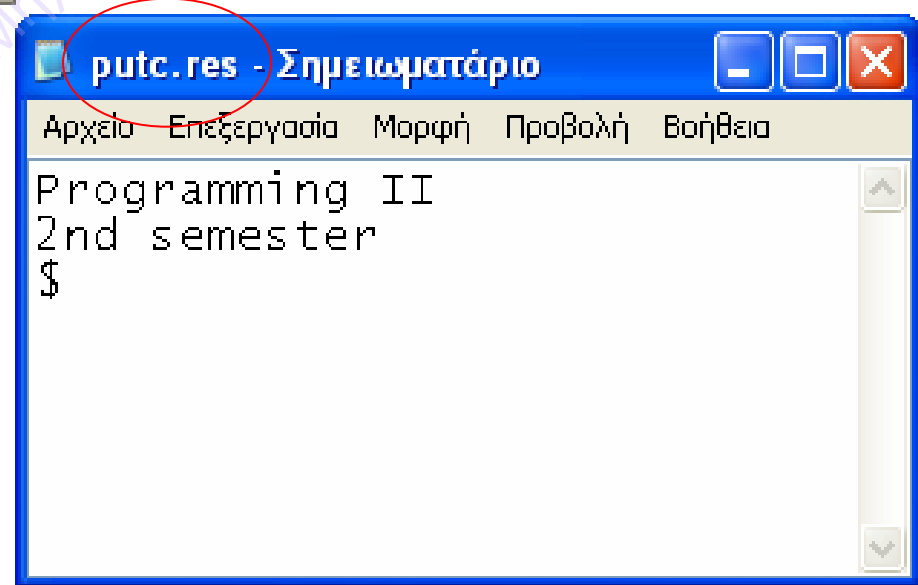
```
    fclose( pF );
```

```
} //end of main
```

Αποτέλεσμα:



```
C:\temp\Project1.exe
Start writting:
Programming II
2nd semester
$
```



```
putc.res - Σημειωματάριο
Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια
Programming II
2nd semester
$
```

Συμπληρωματικό παράδειγμα: Να καταστρωθεί πρόγραμμα , το οποίο διαβάσει οποιοδήποτε αρχείο ASCII και εμφανίζει το περιεχόμενό του στην οθόνη.

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

```
main() {
```

```
    FILE *pF;
```

```
    char ch;
```

```
    pF=fopen( "putc.res","r" ); //Produced in the example of putc
```

```
    if (pF==NULL) printf( "\t\tFILE ERROR: Exit program\n" );
```

```
    else {
```

```
        printf( "\t\tPRESS ANY KEY TO START\n" ); ch=getc(pF);
```

```
        while (ch!=EOF) {
```

```
            putchar(ch);
```

```
            ch=getc(pF);
```

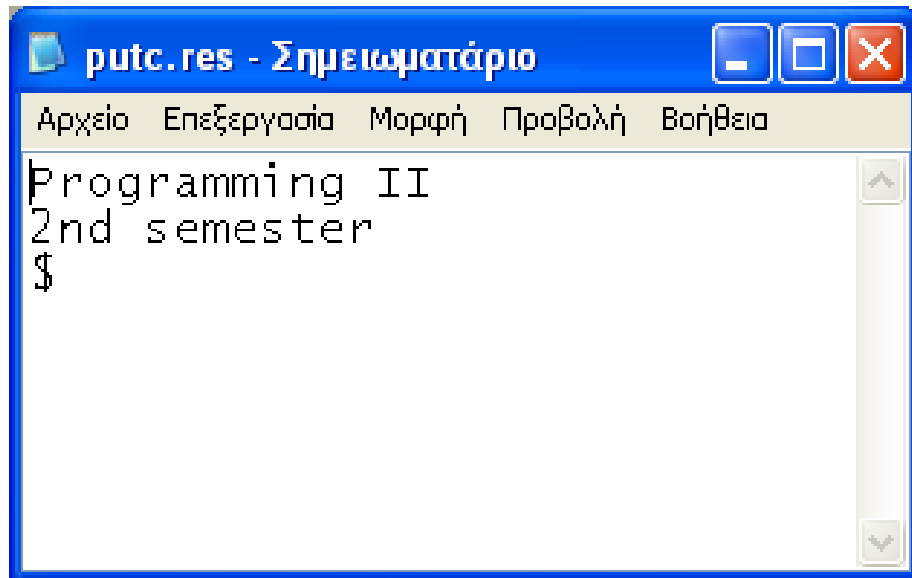
```
        } //end of while
```

```
    } //end of else
```

```
    fclose( pF );
```

```
} //end of main
```

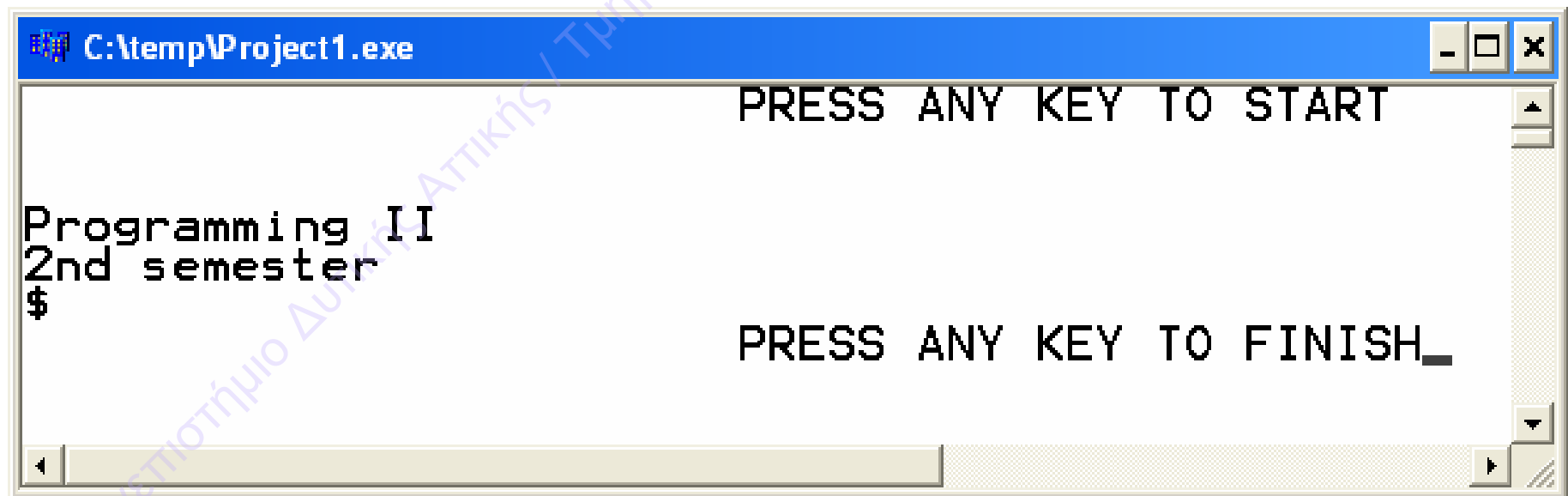
Αποτέλεσμα:



putc.res - Σημειωματάριο

Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια

```
Programming II  
2nd semester  
$
```



C:\temp\Project1.exe

```
PRESS ANY KEY TO START  
  
Programming II  
2nd semester  
$  
  
PRESS ANY KEY TO FINISH_
```

Θεματική ενότητα 10: **Δυναμική διαχείριση μνήμης**

- Ένας πίνακας ελέγχει ένα **συγκεκριμένο μπλοκ μνήμης**.
- Το μέγεθος ενός πίνακα είναι **σταθερό: καθορισμένο** όταν το πρόγραμμα έχει γραφεί.

```
int array[4];
```

- Οι δείκτες μπορούν να **δείξουν** σε μπλοκ μνήμης αλλά . . .
- πώς μπορούν να ελέγξουν **τα δικά τους μπλοκ μνήμης**;
- Τι συμβαίνει όταν πρέπει να **μεταβάλλεται το μέγεθος του πίνακα**, ή θέλουμε να **επιλέγεται το μέγεθος του πίνακα** μετά την έναρξη εκτέλεσης του προγράμματος; ('run time')

- Απάντηση: **δυναμική εκχώρηση μνήμης** (memory allocation) = εκχώρηση μνήμης κατά τη διάρκεια της εκτέλεσης του προγράμματος.
- Επιτρέπει:
 - Τον καθορισμό του μεγέθους κατά την εκτέλεση (συνάρτηση: **malloc()**)
 - Αλλαγή του μεγέθους κατά την εκτέλεση (συνάρτηση: **realloc()**)

void * malloc (int size);

malloc(): επιστρέφει ένα δείκτη στην αρχή του μπλοκ, στο οποίο στο οποίο γίνεται η εκχώρηση. Πρέπει ΠΑΝΤΟΤΕ να γίνεται μετατροπή τύπου έτσι ώστε ο τύπος του δείκτη να είναι ίδιος με τα στοιχεία στα οποία δείχνει.

Η **malloc()** ορίζεται στο **stdlib.h** ή στο **alloc.h**

Ο αριθμός των bytes που θα εκχωρηθούν. Χρησιμοποιείτε τη **sizeof()** για να βρείτε το μέγεθος ενός τύπου.

void free (void *);

Η **free()** δεν επιστρέφει τίποτε. Απλώς αποδεσμεύει τη μνήμη που είχε εκχωρηθεί από τη **malloc()**. Δηλαδή αποδεσμεύει μνήμη έτσι ώστε η τελευταία να μπορεί να χρησιμοποιηθεί αλλού.

Η συνάρτηση **free()** ορίζεται στο **stdlib.h** ; ή στο **alloc.h**.
ΠΡΟΣΟΧΗ: Οι **malloc()** και **free()** αναγκάζουν η μία την άλλη. Εάν χρησιμοποιηθεί η **malloc()** για εκχώρηση μνήμης, πρέπει να χρησιμοποιηθεί η **free()** για την αποδέσμευσή της.

Δείκτης στην αρχή του μπλοκ που θέλουμε να αποδεσμευθεί.

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

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

← Οι **malloc** και **free** ορίζονται στο **stdlib.h**

```
main ( ) {
```

```
    int *start, size;
```

```
    printf("Type the size: ");
```

```
    scanf("%d", &size);
```

```
    start = (int *) malloc (size * sizeof(int));
```

```
    /* other operations here */
```

```
    free(start); /* more on that later */
```

```
}
```

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

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

```
main ( ) {
```

```
    int *start, size; ← Δείκτης σε int  
                      ← Ακέραια μεταβλητή
```

```
    printf("Type the size: ");
```

```
    scanf("%d", &size);
```

```
    start = (int *) malloc (size * sizeof(int));
```

```
    /* other operations here */
```

```
    free(start); /* more on that later */
```

```
}
```

| | |
|------|-------------|
| 900: | start, junk |
| 904: | size, junk |
| 908: | |
| 912: | |
| 916: | |
| 920: | |
| 924: | |

```
#include<stdio.h>
#include<stdlib.h>
main ( ) {
    int *start, size;
    printf("Type the size: ");
    scanf("%d", &size);
    start = (int *) malloc (size * sizeof(int));
    /* other operations here */
    free(start); /* more on that later */
}
```

Θεωρούμε ότι ο
χρήστης ορίζει το
μέγεθος ίσο με 3

| | |
|------|--------------------|
| 900: | start, <i>junk</i> |
| 904: | size, 3 |
| 908: | |
| 912: | |
| 916: | |
| 920: | |
| 924: | |

Η **malloc** αναζητά ένα συνεχές μπλοκ 12 bytes και, όταν το βρει, επιστρέφει ένα δείκτη στην αρχή του μπλοκ. Δηλαδή, **επιστρέφει τη διεύθυνση** του πρώτου byte σ' αυτό το μπλοκ. Η διεύθυνση αυτή ανατίθεται στο δείκτη **start**.

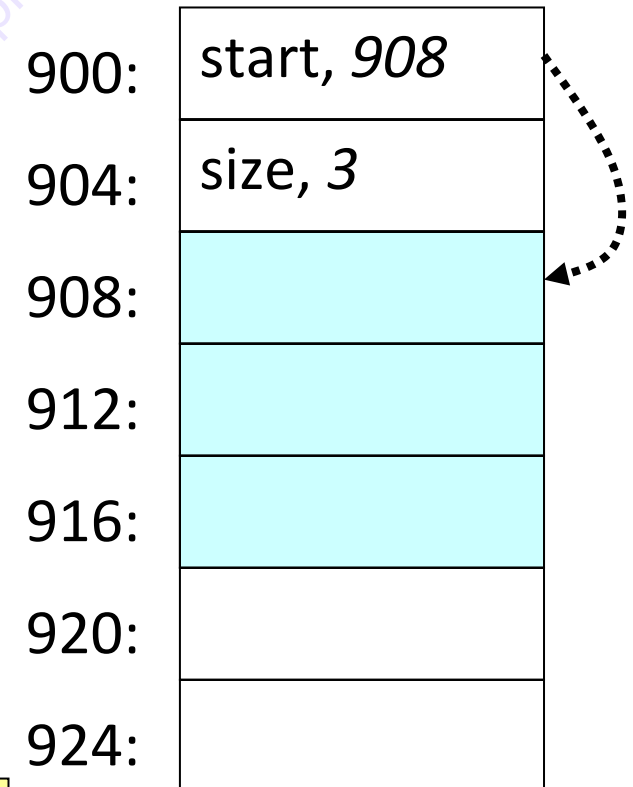
```
scanf("%d", &size),
```

```
start = (int *) malloc (size * sizeof(int));
```

```
/* other operations here */
```

```
free(start); /* more on that later */
```

Η **sizeof** δέχεται ως είσοδο ένα όνομα τύπου και επιστρέφει το μέγεθος του τύπου αυτού, π.χ. ένας ακέραιος έχει μέγεθος 4 bytes.



```
#include<stdio.h>
#include<stdlib.h>
main ( ) {
    int *start = 15;
    *(start+1) = 28;
    *(start+2) = *(start+1) + 12;
    start = (int *) malloc (size * sizeof(int));
    /* other operations here */
    free(start); /* more on that later */
}
```

| | |
|------|------------|
| 900: | start, 908 |
| 904: | size, 3 |
| 908: | 15 |
| 912: | 28 |
| 916: | 40 |
| 920: | |
| 924: | |


```
#include<stdio.h>
#include<stdlib.h>
void ( ) {
    int *start, size;
    printf("Type the size: ");
    scanf("%d", &size);
    start = (int *) malloc (size * sizeof(int));
    /* other operations here */
    free(start);
}
```

| | |
|------|-------------|
| 900: | start, junk |
| 904: | size, 3 |
| 908: | |
| 912: | |
| 916: | |
| 920: | |
| 924: | |

Απελευθερώνει το μπλοκ μνήμης, το οποίο αρχίζει από τη διεύθυνση που καθορίζεται στην αρχή

malloc_second_example.c

malloc

- Τι ΔΕΝ πρέπει να κάνετε:

```
int *pscores;  
pscores = malloc (32);
```



ΛΑΘΟΣ!!!

Χαμένη μετατροπή τύπου

Αν και μερικοί μεταγλωττιστές μπορεί να «καταλάβουν» τι συμβαίνει, πρέπει πάντοτε να θεωρείται ότι δεν μπορούν

Συγκεκριμένος αριθμός bytes.

Δεν ξεκαθαρίζει πώς πολλοί ακέραιοι θα ενταχθούν σ' αυτό το μπλοκ. Επειδή δεν εκχωρούν όλα τα μηχανήματα 4 bytes για int, μην κάνετε υποθέσεις.

- Γιατί η **malloc** δεν μπορεί να βρει τη ζητηθείσα μνήμη;
- Επιστρέφει **NULL**.
- Το **NULL** είναι η διεύθυνση 0.
- Είναι έγκυρη διεύθυνση, που **εγγυημένα δεν περιέχει ποτέ έγκυρα δεδομένα**.
- Καλή προγραμματιστική πρακτική: Να ελέγχετε πάντοτε κατά πόσον η **malloc** επιστρέφει **NULL**:

```
char *pmessage;  
pmessage = (char *) malloc (20 * sizeof(char));  
if (pmessage == NULL) {  
    printf("Insufficient memory. Exiting...");  
    return -1;  
}
```

free

- Η ***free()*** δέχεται ως όρισμα ένα δείκτη, ο οποίος δείχνει στην αρχή του μπλοκ που απελευθερώνεται.
- Δε χρειάζεται να είναι ο ίδιος δείκτης που χρησιμοποιήθηκε στη ***malloc***.
- Το ακόλουθο είναι σωστό:

```
char *pmessage, *msg, aLetter;  
pmessage = (char *) malloc (20 * sizeof(char));  
/* do stuff with pmessage */  
msg = pmessage; /* πλέον και οι δύο δείχνουν στην ίδια θέση */  
pmessage = &aLetter; /* πλέον ο pmessage δείχνει στο aLetter */  
free(msg);
```

- **ΠΡΟΣΟΧΗ:** Μην προσπαθήσετε να απελευθερώσετε την ίδια μνήμη δύο φορές!!
- Το ακόλουθο είναι **ΛΑΘΟΣ**:

```
char *pmessage, *msg, aLetter;  
pmessage = (char *) malloc (20 * sizeof(char));  
/* do stuff with pmessage */  
msg = pmessage; /* πλέον και οι δύο δείχνουν στην ίδια θέση */  
free(msg);  
free(pmessage); /* ΛΑΘΟΣ: Το μπλοκ έχει ήδη απελευθερωθεί!! */
```

malloc_second_example.c

chapter_8_exercise_2.c

chapter_8_exercise_5.c

chapter_8_exercise_6.c

example_struct_file_function_6.c

example_malloc_file_call by value.c

example_malloc_file_call by reference.c

ΔΕΙΚΤΕΣ ΣΕ ΔΕΙΚΤΕΣ

Τα μεγέθη των πινάκων ρυθμίζονται με **δυναμική διαχείριση μνήμης**:

```
main()
{
  char **name; /* pointer-to-(pointers-to char) */
  int i;
  name = (char **)malloc(3*sizeof(char *));
  for (i=0;i<3;i++)
    name[i]=(char *)malloc(40*sizeof(char));
  name[0] = "Zero";
  name[1] = "One";
  name[2] = "Two";
  printf("%s,%s,%s,",name[0],name[1],name[2]);
  printf("%c,%c,%c!\n",
    name[0][0],name[1][0],name[2][0]);

  free(name);
}
```

Δήλωση δείκτη για λίστα ...

Αποτέλεσμα: Zero,One,Two,Z,O,T!

ΔΕΙΚΤΕΣ ΣΕ ΔΕΙΚΤΕΣ

Τα μεγέθη των πινάκων ρυθμίζονται με **δυναμική διαχείριση μνήμης**:

```
main()
{
    char **name; /* pointer-to-(pointers-to char)* */
    int i;
    name = (char **)malloc(3*sizeof(char *));
    for (i=0;i<3;i++)
        name[i]=(char *)malloc(40*sizeof(char));
    name[0] = "Zero";
    name[1] = "One";
    name[2] = "Two";
    printf("%s,%s,%s,",name[0],name[1],name[2]);
    printf("%c,%c,%c!\n",
        name[0][0],name[1][0],name[2][0]);

    free(name);
}
```

Δήλωση δείκτη για λίστα ...

Αποτέλεσμα: Zero,One,Two,Z,O,T!

ΔΕΙΚΤΕΣ ΣΕ ΔΕΙΚΤΕΣ

Τα μεγέθη των πινάκων ρυθμίζονται με **δυναμική διαχείριση μνήμης**:

```
main()
{
    char **name; /* pointer-to-(pointers-to char) */
    int i;
    name = (char **)malloc(3*sizeof(char *));
    for (i=0;i<3;i++)
        name[i]=(char *)malloc(40*sizeof(char));
    name[0] = "Zero";
    name[1] = "One";
    name[2] = "Two";
    printf("%s,%s,%s,",name[0],name[1],name[2]);
    printf("%c,%c,%c!\n",
        name[0][0],name[1][0],name[2][0]);

    free(name);
}
```

Χώρος για 3 (pointers-to-char)

Αποτέλεσμα: Zero,One,Two,Z,O,T!

ΔΕΙΚΤΕΣ ΣΕ ΔΕΙΚΤΕΣ

Τα μεγέθη των πινάκων ρυθμίζονται με **δυναμική διαχείριση μνήμης**:

```
main()
{
    char **name; /* pointer-to-(pointers-to char) */
    int i;
    name = (char **)malloc(3*sizeof(char *));
    for (i=0;i<3;i++)
        name[i]=(char *)malloc(40*sizeof(char));
    name[0] = "Zero";
    name[1] = "One";
    name[2] = "Two";
    printf("%s,%s,%s,",name[0],name[1],name[2]);
    printf("%c,%c,%c!\n",
        name[0][0],name[1][0],name[2][0]);

    free(name);
}
```

Χώρος για 40 χαρακτήρες

Δεν απαιτείται *strcpy*

Αποτέλεσμα: Zero,One,Two,Z,O,T!

ΔΕΙΚΤΕΣ ΣΕ ΔΕΙΚΤΕΣ

Τα μεγέθη των πινάκων ρυθμίζονται με **δυναμική διαχείριση μνήμης**:

```
main()
{
char **name; /* pointer-to-(pointers-to char) */
int i;
name = (char **)malloc(3*sizeof(char *));
for (i=0;i<3;i++)
    name[i]=(char *)malloc(40*sizeof(char));
name[0] = "Zero";
name[1] = "One";
name[2] = "Two";
printf("%s,%s,%s,", name[0], name[1], name[2]);
printf("%c,%c,%c!\n",
        name[0][0], name[1][0], name[2][0]);

free(name);
}
```

Χρησιμοποιεί τον **δείκτη** του πίνακα για να επιλέξει αλφαριθμητικό

Αποτέλεσμα: **Zero,One,Two,Z,O,T!**

Δείκτες σε δείκτες

Τα μεγέθη των πινάκων ρυθμίζονται με **δυναμική διαχείριση μνήμης**:

```
main()
{
char **name; /* pointer-to-(pointers-to char) */
int i;
name = (char **)malloc(3*sizeof(char *));
for (i=0;i<3;i++)
    name[i]=(char *)malloc(40*sizeof(char));
name[0] = "Zero";
name[1] = "One";
name[2] = "Two";
printf("%s,%s,%s,",name[0],name[1],name[2]);
printf("%c,%c,%c!\n",
    name[0][0],name[1][0],name[2][0]);

free(name);
}
```

Ο δεύτερος δείκτης επιλέγει **χαρακτήρες** στο **αλφαριθμητικό**

Αποτέλεσμα: Zero,One,Two,**Z,O,T**!

ΔΕΙΚΤΕΣ ΣΕ ΔΕΙΚΤΕΣ

Τα μεγέθη των πινάκων ρυθμίζονται με **δυναμική διαχείριση μνήμης**:

(το *name* δείχνει σε δεσμευμένη μνήμη
κι όχι το ***name* !!)

Απελευθερώνει τη μνήμη
στο τέλος

```
char **name; /* pointer-to-(pointers-to char) */
int i;
name = (char **)malloc(3*sizeof(char *));
for (i=0;i<3;i++)
    name[i]=(char *)malloc(40*sizeof(char));
name[0] = "Zero";
name[1] = "One";
name[2] = "Two";
printf("%s,%s,%s,",name[0],name[1],name[2]);
printf("%c,%c,%c!\n",
    name[0][0],name[1][0],name[2][0]);
free(name);
}
```

Αποτέλεσμα: Zero,One,Two,Z,O,T!

Δείκτες σε δείκτες

Τι συμβαίνει:

```
name = (char **)malloc(3*sizeof(char *));
```

«**Δέσμευσε ένα μπλοκ μνήμης**, επαρκές για 3 δείκτες **χαρακτήρα**. Στη συνέχεια να επιστρέψεις ένα δείκτη σ' αυτό».

(δείκτης σε λίστα **δεικτών** χαρακτήρα)

name

... **name[0]** **name[1]** **name[2]** ...

(Κάθε **στοιχείο** είναι ένας δείκτης χαρακτήρα. Ο μηδενικός δείκτης είναι το 'name[0]', ο πρώτος δείκτης είναι το 'name[1]', κ.λ.π.)

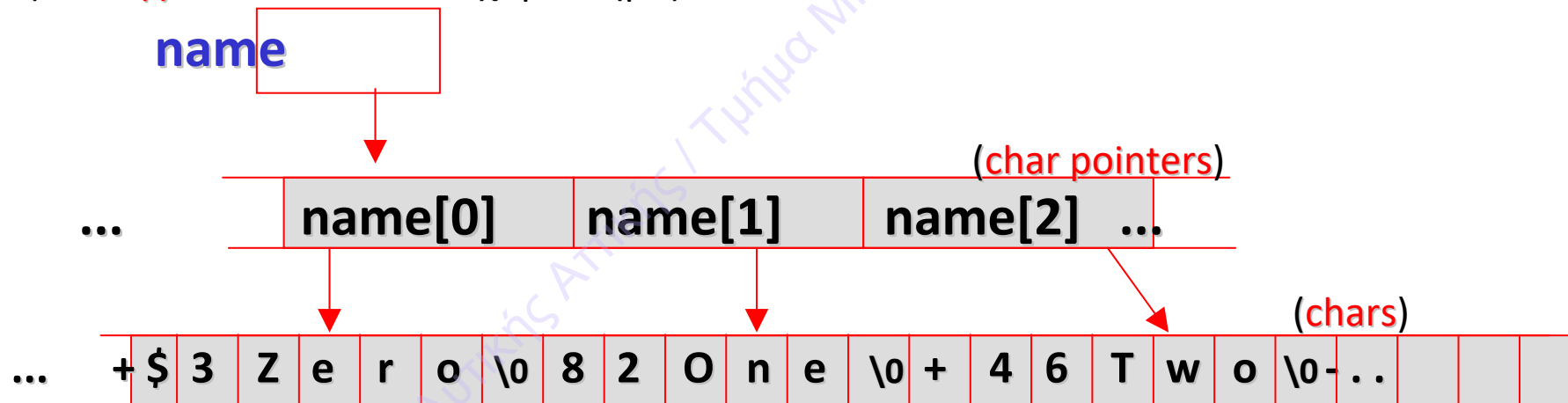
Δείκτες σε δείκτες

Τι συμβαίνει:

```
name[0] = "Zero";  
name[1] = "One";  
name[2] = "Two";
```

(δείκτης σε λίστα δεικτών χαρακτήρα)

name



«Κάθε **δείκτης** της λίστας πρέπει να δείχνει σε αλφαριθμητική σταθερά που είναι αποθηκευμένη στη μνήμη»

Δείκτες σε δείκτες

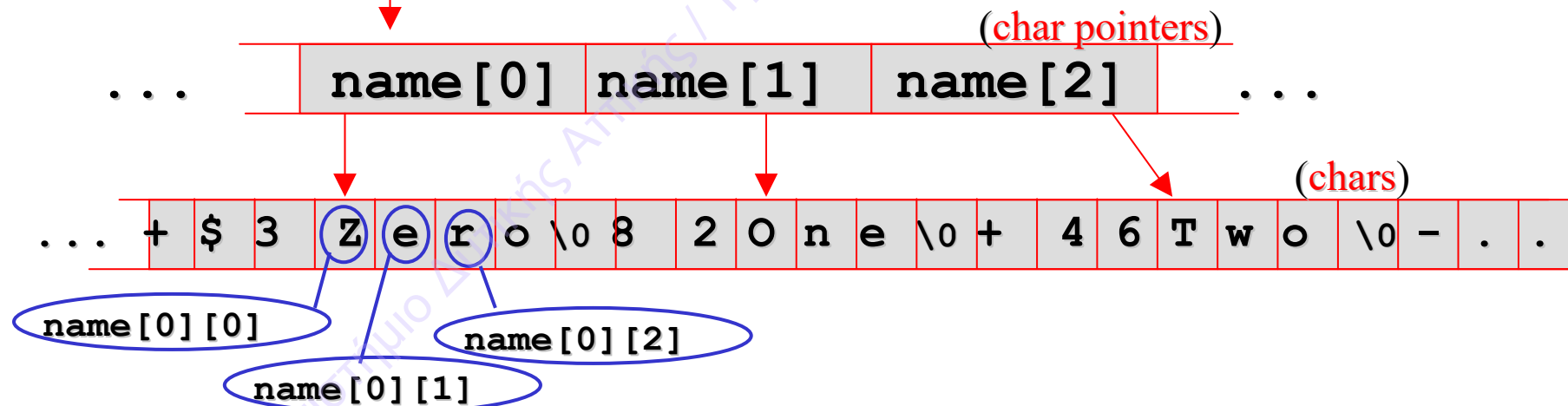
Τι συμβαίνει:

```
name[0] = "Zero";  
name[1] = "One";  
name[2] = "Two";
```

(δείκτης σε λίστα δεικτών χαρακτήρα)

name

«Κάθε **δείκτης** της λίστας πρέπει να δείχνει σε αλφαριθμητική σταθερά που είναι αποθηκευμένη στη μνήμη»



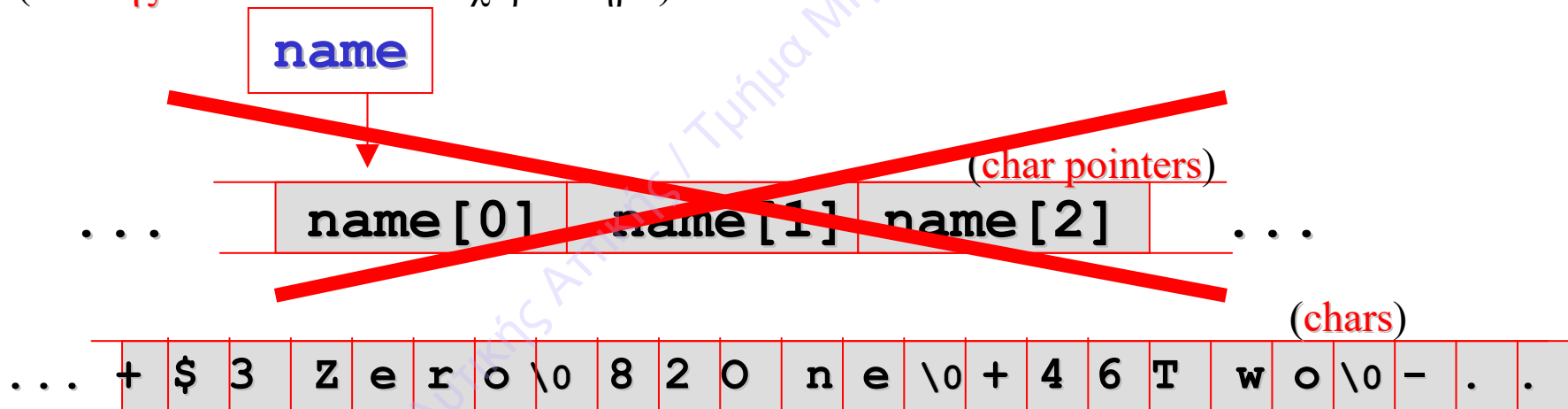
Δείκτες σε δείκτες

Τι συμβαίνει:

```
free (name) ;
```

«Απελευθέρωσε το μπλοκ μνήμης που είχε δεσμευτεί με το δείκτη **'name'**»

(δείκτης σε λίστα δεικτών χαρακτήρα)



ΔΕΙΚΤΕΣ ΣΕ ΔΕΙΚΤΕΣ

Παράδειγμα:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
main() {
    char **name;
    int no_strings,string_size,size,i,j;
    printf("\nGive the number of strings: ");
    scanf("%d",&no_strings);
    printf("\nGive the string size: ");
    scanf("%d",&string_size); // '\0' included
    name=(char **)malloc(no_strings*sizeof(char *));
    for (i=0;i<no_strings;i++)
        name[i]=(char *)malloc(string_size*sizeof(char));
    printf("\naddr(name)=%d name=%d\n",&name,name);
```



```
for (i=0;i<no_strings;i++) {  
    printf("\naddr(name[%d])=%d",i,&name[i]);  
    printf("\n\taddr of first element of  
           name[%d][0]=%d\n",i,&name[i][0]);  
    printf("\n\taddr of last element of name[%d][%d]=%d\n",  
           i,string_size-1,&name[i][string_size-1]);  
}  
for (i=0;i<no_strings;i++)  
{  
    printf("\nGive the %d string: ",i+1);  
    scanf("%s",name[i]);  
}  
for (i=0;i<no_strings;i++)  
{  
    printf("\nname[%d][0])=%c name[%d]=",i,name[i][0],i,name[i]);  
    puts(name[i]);  
}  
free(name);  
system("pause");  
}
```



```
C:\temp\Project1.exe

Give the number of strings: 3
Give the string size: 100
addr(name)=1245064 name=13586172
addr(name[0])=13586172
  addr of first element of name[0][0]=13586188
  addr of last element of name[0][99]=13586287
addr(name[1])=13586176,
  addr of first element of name[1][0]=13586292
  addr of last element of name[1][99]=13586391
addr(name[2])=13586180,
  addr of first element of name[2][0]=13586396
  addr of last element of name[2][99]=13586495

Give the 1 string: PROGRAMMING
Give the 2 string: POINTERS
Give the 3 string: ALLOCATION
name[0][0])=P name[0]=PROGRAMMING
name[1][0])=P name[1]=POINTERS
name[2][0])=A name[2]=ALLOCATION
Πιέστε ένα πλήκτρο για συνέχεια. . .
```

Διαφορετικές θέσεις μνήμης. Το name στη στοίβα και τα υπόλοιπα στο σωρό.

Εάν δηλωθεί πίνακας με τον κλασσικό τρόπο θα αποθηκευθεί στην στοίβα. Παράδειγμα με μεγάλο πίνακα και stack overflow (1 τραγούδι 3' με ποιότητα cd – 44.1 KHz).

Δείκτες σε δείκτες

Πολυδιάστατοι πίνακες float:

```
main()
```

```
{
```

```
float **grid; /* pointer-to-pointer-to-float */
```

```
int i,rmax,cmax;
```

```
grid = (float **)malloc(rmax*sizeof(float *));
```

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

```
{
```

```
    grid[i]=(float *)malloc(cmax*sizeof(float));
```

```
}
```

```
... COMPUTE! ... use grid[r][c] ...
```

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

```
{
```

```
    free(grid[i]);
```

```
}
```

```
free(grid);
```

```
}
```

Δήλωσε δείκτη σε λίστα δεικτών

Δείκτες σε δείκτες

```
main()
{
float **grid; /* pointer-to-pointer-to-float */
int i,rmax,cmax;

grid = (float **)malloc(rmax*sizeof(float *));
for(i=0; i<rmax; i++)
{
grid[i]=(float *)malloc(cmax*sizeof(float));
}
... COMPUTE! ... use grid[r][c] ...
for(i=0; i<rmax; i++)
{
free(grid[i]);
}
free(grid);
}
```

Δημιουργεί ένα μπλοκ
rmax pointers-to-float



Δείκτες σε δείκτες

```
main()
{
float **grid; /* pointer-to-pointer-to-float */
int i,rmax,cmax;

grid = (float **)malloc(rmax*sizeof(float *));
for(i=0; i<rmax; i++)
{
    grid[i]=(float *)malloc(cmax*sizeof(float));
}
... COMPUTE! ... use grid[r][c] ...
for(i=0; i<rmax; i++)
{
    free(grid[i]);
}
free(grid);
}
```

Δημιουργεί ένα μπλοκ
από cmax floats για
κάθε pointer-to-float

Δείκτες σε δείκτες

```
main()
{
float **grid; /* pointer-to-pointer-to-float */
int i,rmax,cmax;

grid = (float **)malloc(rmax*sizeof(float *));
for(i=0; i<rmax; i++)
{
    grid[i]=(float *)malloc(cmax*sizeof(float));
}
... COMPUTE! ... use grid[r][c] ...
for(i=(rmax-1); i>=0; i--)
{
    free(grid[i]);
}
free(grid);
}
```

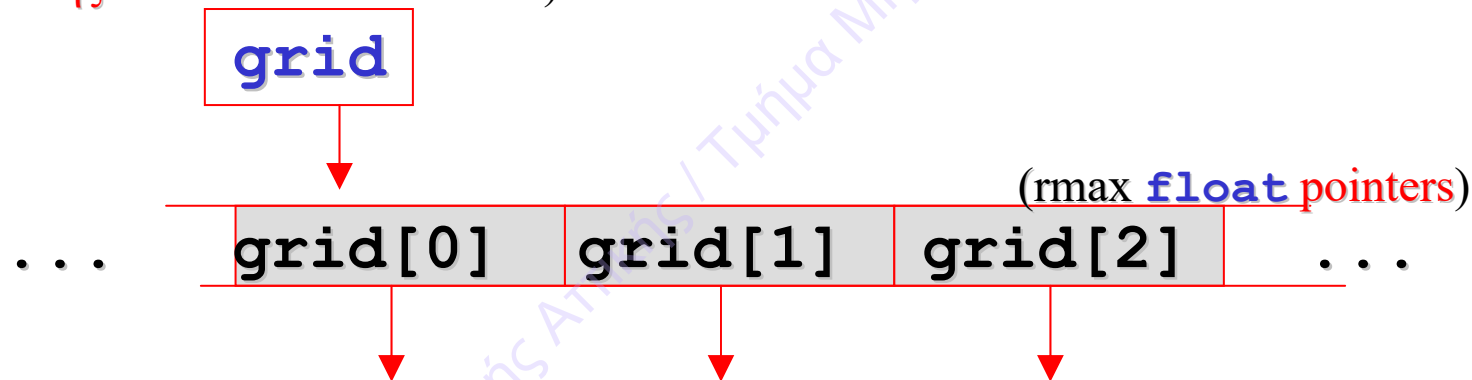
← Τέλος; Να αντιστραφεί η διαδικασία: αρχικά να απελευθερωθεί κάθε μπλοκ από floats και στη συνέχεια κάθε μπλοκ από pointers-to-float

Δείκτες σε δείκτες

```
grid = (float **)malloc(rmax*sizeof(float *));
```

«Δέσμευσε μπλοκ μνήμης για
rmax pointers-to-float»

(δείκτης σε λίστα δεικτών float)



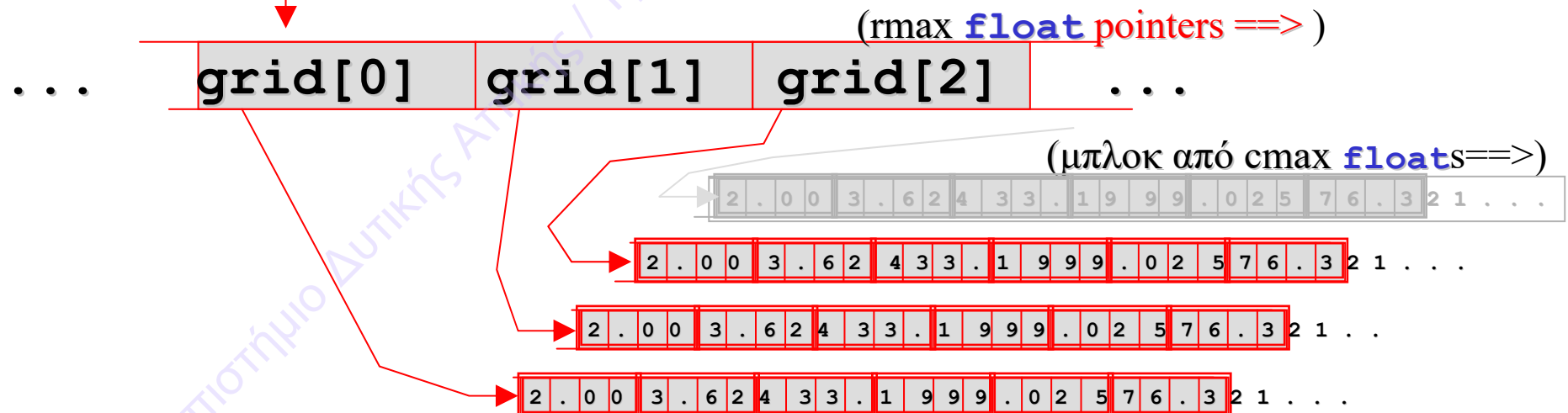
Δείκτες σε δείκτες

```
grid = (float **)malloc(rmax*sizeof(float *));
for(i=0; i<rmax; i++)
{
    grid[i]=(float *)malloc(cmax*sizeof(float));
}
```

(δείκτης σε λίστα δεικτών float)

grid

“Για κάθε δείκτη δέσμευσε ένα μπλοκ μνήμης για cmax floats”



Δείκτες σε δείκτες

`grid[1][2] = radius*(sin(x) + cos(y)...);`

“Προσπέλασε τον x-στό δείκτη και πάρε την y-στή float τιμή του”

(δείκτης σε λίστα δεικτών float)

`grid`

(xmax `float` pointers ==>)

... `grid[0]` `grid[1]` `grid[2]` ...

(μπλοκ από cmax `floats`==>)

