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’)

- **<u>Απάντηση:</u>** ***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. Χρησιμοποιείστε 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()* αναγκαιούν 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>
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 */

}
```

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

```
#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 */

}
```

Δείκτης σε int

Ακέραια μεταβλητή

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 */

}
```

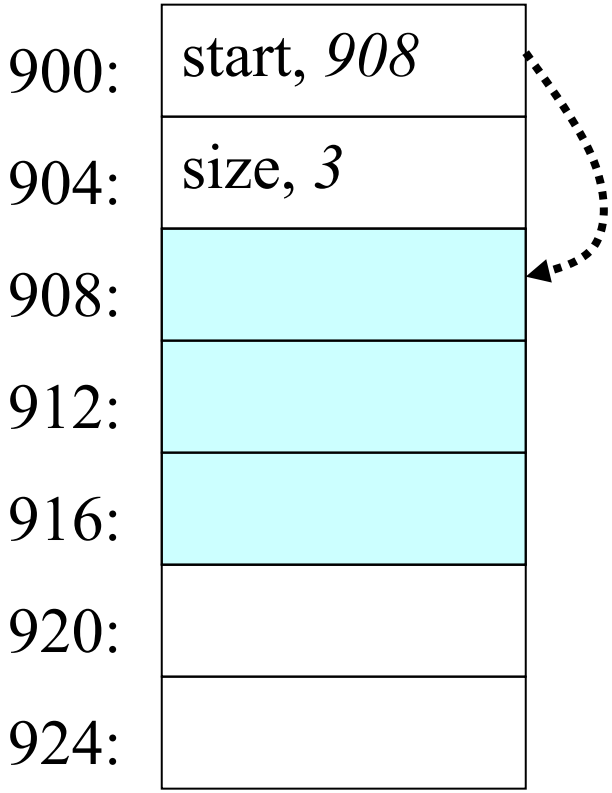
Θεωρούμε ότι ο χρήστης ορίζει το μέγεθος ίσο με 3

900:	start, *junk*
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***.

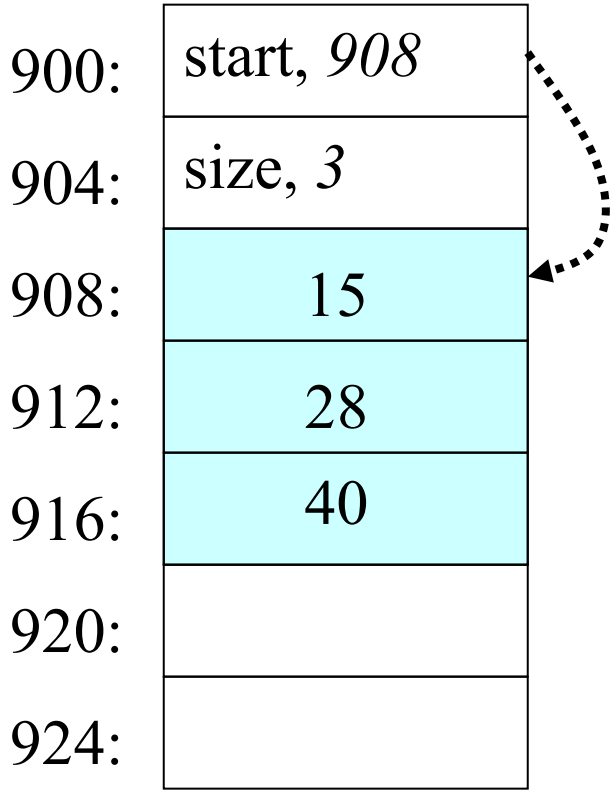
```
scanf("%d", &size);
start = (int *) malloc (size * sizeof(int));
/* other operations here */
free(start); /* more on that later */
}
```

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
    pri
    sca
    start = (int *) malloc (si        sizeof(int));
    /* other operations here */
    free(start); /* more on that later */

}
```

```
*start = 15;
*(start+1) = 28;
*(start+2) = *(start+1) + 12;
```



```
#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 που αρχίζει από τη διεύθυνση που καθορίζεται στην αρχή

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 μηχανή-ματα 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!! */
```

Παρατηρήσεις για malloc(), free()

- Το *free(ptr)* είναι **επικίνδυνο** εάν ο **ptr** δεν είναι έγκυρος!! Το αποτέλεσμα είναι απρόβλεπτο: μπορεί να μη συμβεί τίποτε είτε να υπάρξει κάποιο σφάλμα είτε ακόμη και να κολλήσει το πρόγραμμα.
- Όταν σε ένα πρόγραμμα έχουμε επαναλαμβανόμενες εκχωρήσεις μνήμης χωρίς τις αντίστοιχες απελευθερώσεις, συμβαίνουν 'διαρροές μνήμης'
  - Το πρόγραμμα αυξάνει συνεχώς καθώς εκτελείται
  - Τελικά, το πρόγραμμα **είτε θα πρέπει να σταματήσει είτε θα κολλήσει.**
  - **Για την αποφυγή των ανωτέρω θα πρέπει:**
    - Βέλτιστη λύση: Κρατείστε τα ζεύγη **malloc/free** στο ίδιο τμήμα κώδικα
    - Καλή λύση: Δημιουργείστε μία κεντρική συνάρτηση, λειτουργούσα ως "δοχείο απορριμάτων".
    - Κακή λύση: Διασκορπισμένες **malloc()** και **free()**

malloc_1st_example.cpp

Δείκτες σε δείκτες

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

Δήλωση **δείκτη** για λίστα …

```
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);
}
```

Αποτέλεσμα: Zero,One,Two,Z,O,T!

Δείκτες σε δείκτες

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

Δήλωση **δείκτη** για λίστα …

```
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);
}
```

Αποτέλεσμα: Zero,One,Two,Z,O,T!

Δείκτες σε δείκτες

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

Χώρος για 3 **(pointers-to-char)**

```
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);
}
```

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);
}
```

Χώρος για 40 χαρακτήρες

Δεν απαιτείται strcpy

Αποτέλεσμα: Zero,One,Two,Z,O,T!

Δείκτες σε δείκτες

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

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

```
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);
}
```

Αποτέλεσμα: Zero,One,Two,Z,O,T!

Δείκτες σε δείκτες

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

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

```
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);
}
```

Αποτέλεσμα: Zero,One,Two,Z,O,T!

Δείκτες σε δείκτες

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!

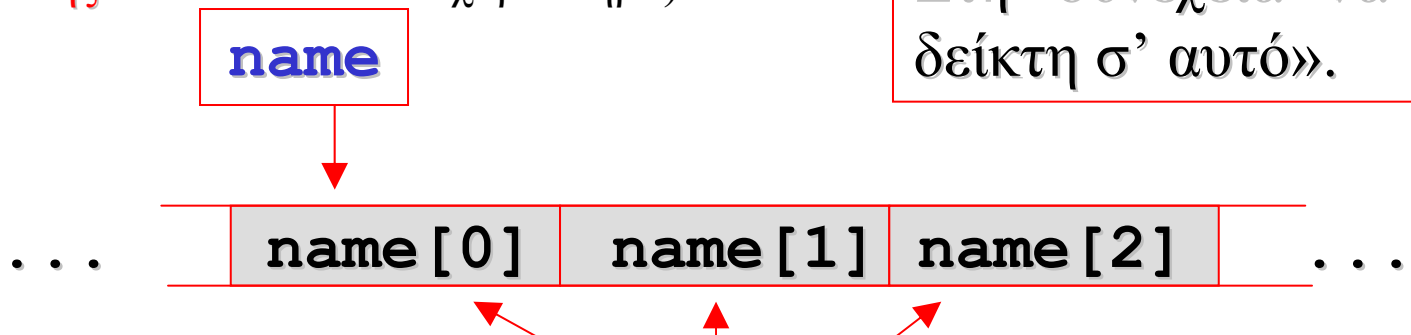
Δείκτες σε δείκτες

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)



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

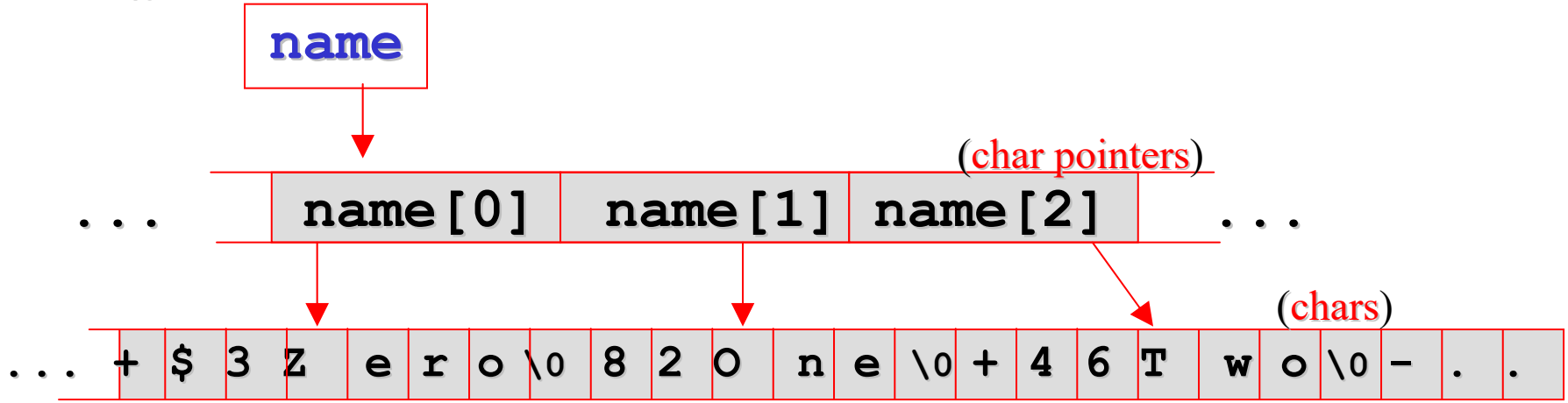
Δείκτες σε δείκτες

Τι συμβαίνει:

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

(δείκτης σε λίστα δεικτών χαρακτήρα)

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



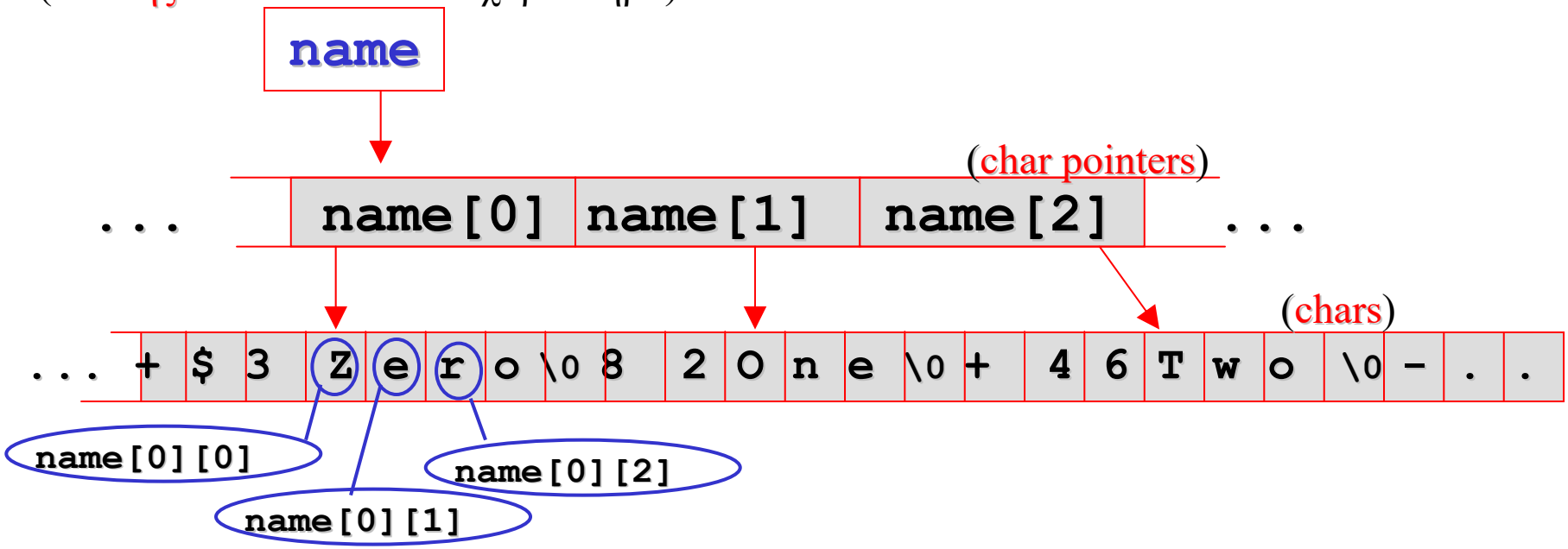
Δείκτες σε δείκτες

Τι συμβαίνει:

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

(δείκτης σε λίστα δεικτών χαρακτήρα)

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



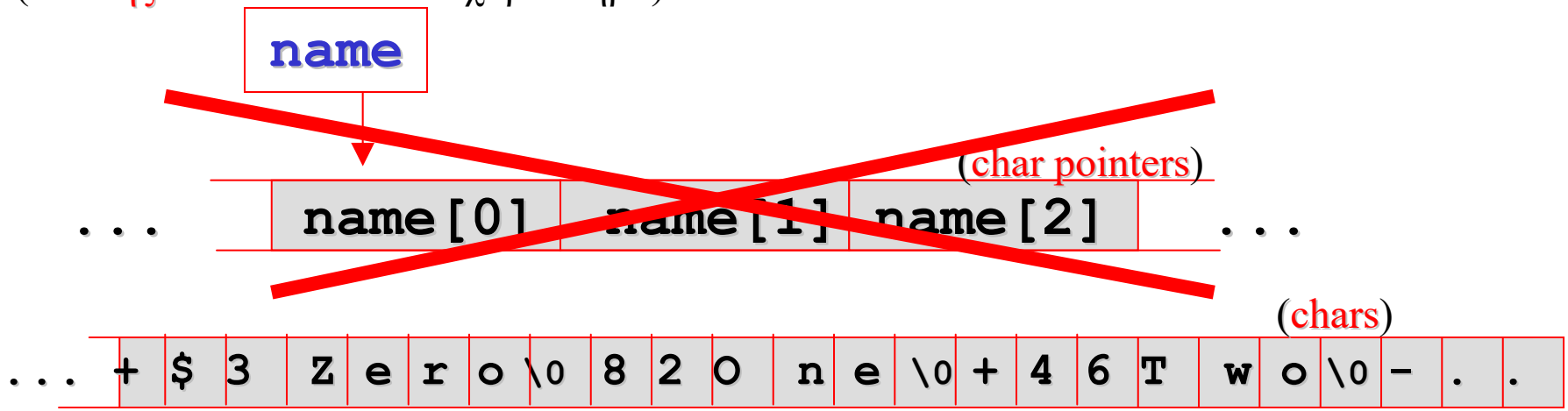
Δείκτες σε δείκτες

Τι συμβαίνει:

```
free(name);
```

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

(δείκτης σε λίστα δεικτών χαρακτήρα)



Δείκτες σε δείκτες

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("\nadrr(name[%d])=%d,",i,&name[i]);
    printf("\n\tadrr of first element of
                                name[%d][0]=%d\n",i,&name[i][0]);
    printf("\n\tadrr 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
adrr(name[0])=13586172,
        adrr of first element of name[0][0]=13586188
        adrr of last element of name[0][99]=13586287
adrr(name[1])=13586176,
        adrr of first element of name[1][0]=13586292
        adrr of last element of name[1][99]=13586391
adrr(name[2])=13586180,
        adrr of first element of name[2][0]=13586396
        adrr 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:

Δήλωσε δείκτη σε λίστα δεικτών

```
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);
}
```

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);
}
```

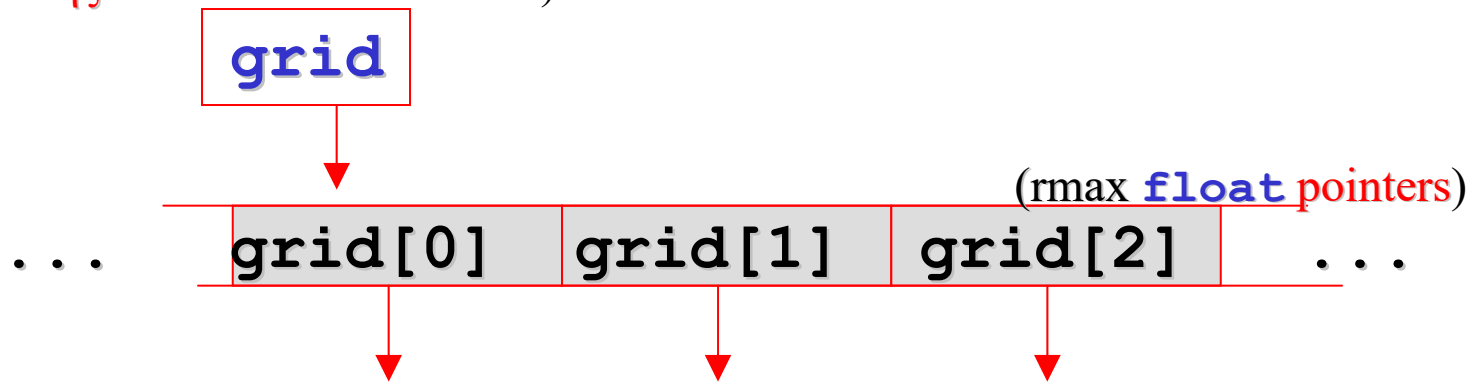
Τέλος; Να αντιστραφεί η διαδικασία: αρχικά να απελευθερωθεί κάθε μπλοκ από floats και στη συνέχεια κάθε μπλοκ από pointers-to-float

Δείκτες σε δείκτες

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

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

(δείκτης σε λίστα δεικτών float)

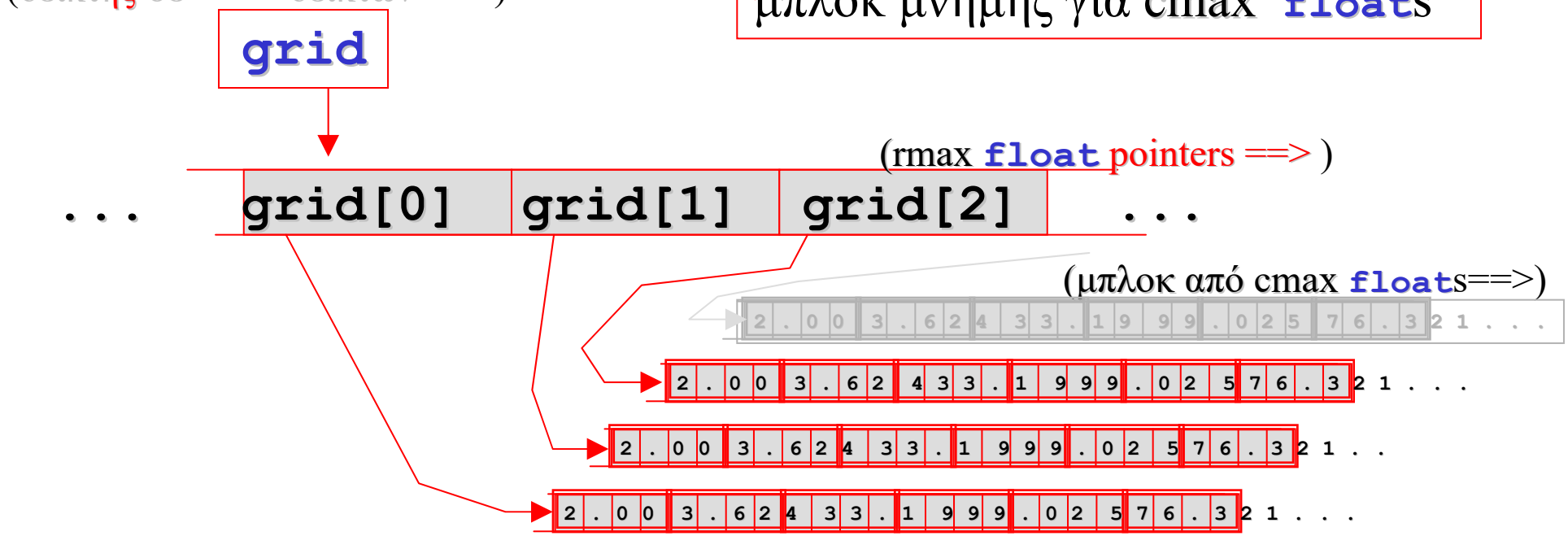


Δείκτες σε δείκτες

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

"Για κάθε δείκτη δέσμευσε ένα μπλοκ μνήμης για cmax floats"

(δείκτης σε λίστα δεικτών float)

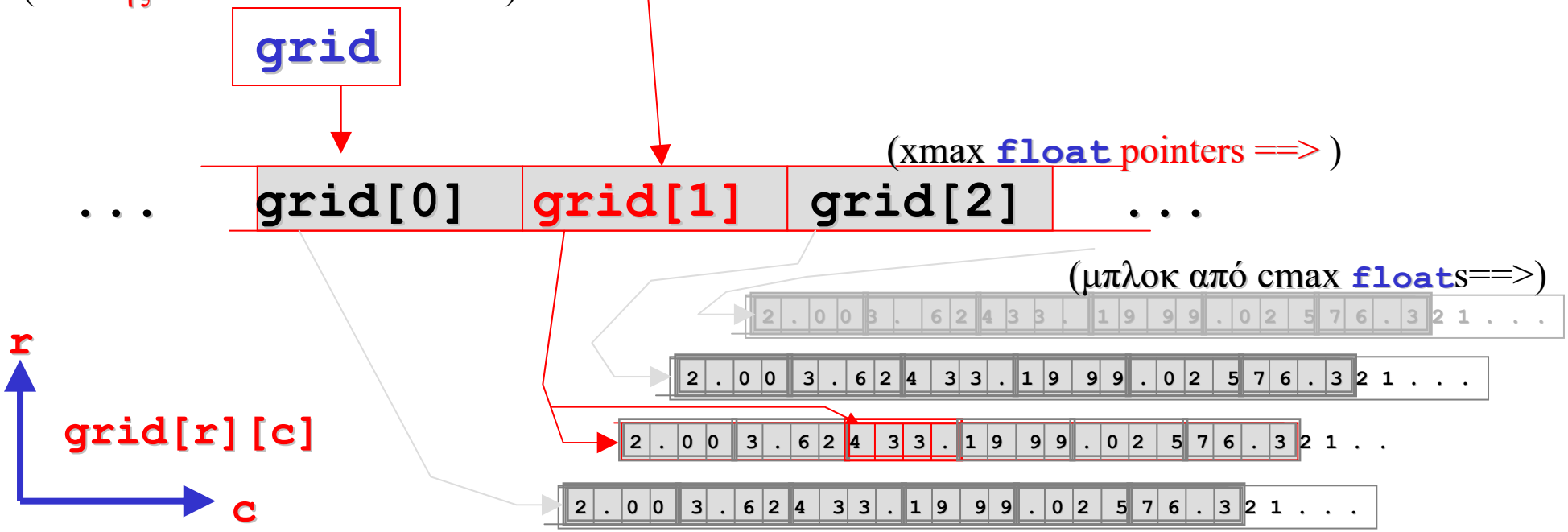


Δείκτες σε δείκτες

```
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)



Έλεγχος επάρκειας μνήμης κατά την εκχώρηση μνήμης

```
x=(float **)malloc(size1*sizeof(float *));
                        assert(x!=NULL);
for (i=0; i<size; i++)
{
  x[i]=(float *)malloc(size*sizeof(float));
                          assert(x[i]!=NULL);
}
```

Καθώς η επιστρεφόμενη τιμή του *x* είναι NULL σε περίπτωση σφάλματος, μπορούμε να χρησιμοποιήσουμε την εντολή *assert* για να πιστοποιήσουμε ότι υπάρχει διαθέσιμη μνήμη. Η συνάρτηση *assert* απαιτεί να συμπεριληφθεί το <*assert.h*>. Σε περίπτωση που δεν υπάρχει διαθέσιμη μνήμη, το πρόγραμμα σταματά και εμφανίζεται το μήνυμα *Assertion failed*, καθώς και γραμμή κώδικα στην οποία εμφανίσθηκε η έλλειψη μνήμης.

Συναρτήσεις οριζόμενες από τον χρήστη για τη δέσμευση/αποδέσμευση μνήμης

- Στην περίπτωση που στο πρόγραμμα γίνεται επανειλημμένα δέσμευση και αποδέσμευση μνήμης, μπορούν να ορισθούν συναρτήσεις που θα δεσμεύουν και θα αποδεσμεύουν μνήμη για πίνακες float, int κ.λ.π.
- Οι συναρτήσεις δέσμευσης μνήμης θα έχουν παραμέτρους τα μεγέθη της μνήμης που ζητείται να δεσμευθεί και θα επιστρέφουν το δείκτη που θα διαχειρίζεται τη μνήμη.
- Οι συναρτήσεις αποδέσμευσης μνήμης θα έχουν παραμέτρους τον δείκτη που διαχειρίσθηκε τη μνήμη και το μέγεθος αυτής. Δε θα έχουν επιστρεφόμενη τιμή.

Συναρτήσεις οριζόμενες από το χρήστη για τη δέσμευση/αποδέσμευση μνήμης

Οι ακόλουθες συναρτήσεις δεσμεύουν/αποδεσμεύουν μνήμη για ένα δισδιάστατο πίνακα αριθμών κινητής υποδιαστολής:

```
float **allocate_2(int size1, int size2) {
  int i;
  float **deikt;
  deikt=(float **)malloc(size1*sizeof(float *));
assert(deikt!=NULL);
  for (i=0;i<size1;i++) {
    deikt[i]=(float *)malloc(size2*sizeof(float));
    assert(deikt[i]!=NULL);
  }
  return(deikt);
}
```

Συναρτήσεις οριζόμενες από το χρήστη για τη δέσμευση/αποδέσμευση μνήμης

```
void free_2(float **deikt, int size1)
{
  int i;
  for (i=(size1-1);i>=0;i--) free(deikt[i]);
  free(deikt);
}
```

Καλούνται ως εξής:

απαιτείται

```
float **s;
s=allocate_2(3,250);//δέσμευση μνήμης για πίνακα s[3][250]
free_2(s,3);//αποδέσμευση μνήμης του πίνακα s[3][250]
```

δεν απαιτείται

C:\temp\Project1.exe

```
deikt=13061472          addr(deikt)=1245040
deikt[0]=13061488               addr(deikt[0])=13061472
deikt[1]=13062492               addr(deikt[1])=13061476
deikt[2]=13063496               addr(deikt[2])=13061480

s=13061472              addr(s)=1245064
s[0]=13061488           addr(s[0])=13061472
s[1]=13062492           addr(s[1])=13061476
s[2]=13063496           addr(s[2])=13061480
(Prior to free)   s[0][0]=400.000000
(afterwards) s[0][0]=0.000000
```

Πίνακας 3 θέσεων με στοιχεία pointers

Αυτόματη τοποθέτηση τιμής. Ουσιαστικά η μνήμη έχει καθαρισθεί.

Example_of_allocation_functions.cpp

(υπενθύμιση) string.h

- Εύρεση μήκους length
- Αντιγραφή ενός string
- Συνένωση 2 strings
- Σύγκριση 2 strings
- Εύρεση χαρακτήρα σε string
- Εύρεση string σε string

Όλοι οι chars	Οι πρώτοι n chars
`strlen()`	
`strcpy()`	`strncpy()`
`strcat()`	`strncat()`
`strcmp()`	`strncmp()`
`strchr()`	`strrchr()`
`strstr()`	

Χρήση δεικτών σε αλφαριθμητικά

- Μπορούν να εκτελεσθούν πράξεις ακεραίων
- Μπορούν να χρησιμοποιηθούν ως κινητά ονόματα πινάκων. Ο δείκτης πίνακα (array index) [ ] λειτουργεί!

*(arr+n) = arr[n]

arr+n = &arr[n]

- Οι δείκτες μπορούν να ορίσουν ένα string όπως ακριβώς το ορίζει ένας πίνακας.

Παράδειγμα: Να δοθεί το πρωτότυπο της συνάρτησης ***strlen()*** (μήκος string).

Λύση: Θεωρούμε ότι το όρισμα που δέχεται η συνάρτηση είναι δείκτης σε χαρακτήρα και η δήλωση διαμορφώνεται ως εξής:

```
int strlen(char *s);
```

Το σώμα της συνάρτησης με δείκτες και με πίνακες δίνεται παρακάτω:

```
int strlen(char *s)
{
   char *p=s;
   while (*s!='\0')
    s++;
   return(s-p);
}
```

```
int strlen(char str[])
{
   int i=0;
   while (str[i++]);
   return(i-1);
}
```

δυσνόητο

Η εντολή `s-p` εκτελεί αφαίρεση δεικτών και δίνει τον αριθμό των στοιχείων μεταξύ των δύο δεικτών.

Example_of_strlen.cpp

Παράδειγμα: Να δοθεί το πρωτότυπο της συνάρτησης ***strcpy()*** (αντιγραφή ενός string σε ένα άλλο).

```
// Έκδοση με πίνακες
void strcpy(char s[], char t[])
{
 int i=0;
 while ((s[i]=t[i])!='\0') i++;
}
```

```
// 1η έκδοση με δείκτες
void strcpy(char *s, char *t)
{
 while ((*s=*t)!='\0'){
  s++;
  t++;
 }
}
```

```
// 2η έκδοση με δείκτες
void strcpy(char *s, char *t)
{
 while ((*s++=*t++)!='\0');
}
```

```
// 3η έκδοση με δείκτες
void strcpy(char *s, char *t)
{
 while (*s++=*t++);
}
```

Οι εκδόσεις είναι ισοδύναμες; Δοκιμάστε σε δικά σας παραδείγματα!!

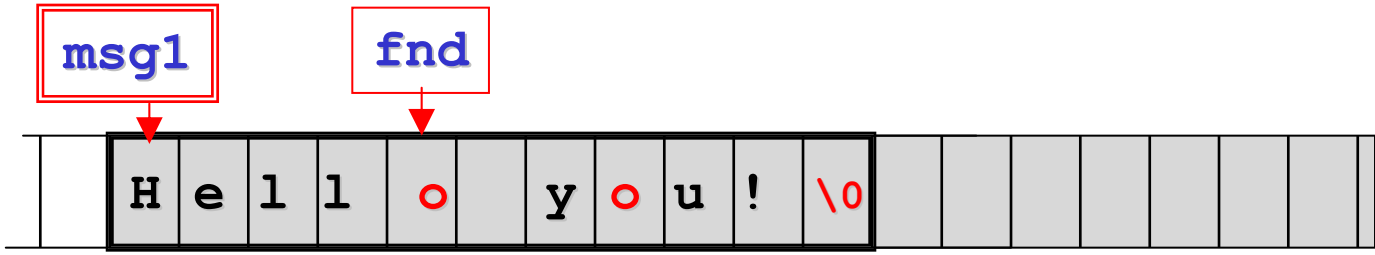
Εύρεση χαρακτήρα: strchr(str1,ch)

```
#include <stdio.h>        /* για την printf()    */
#include <string.h>       /* για την strcpy()    */

main()
{
   char msg1[81]={"Hello you!"};
   char *fnd;             /* pointer to char */


   fnd = strchr(msg1,'o');
   printf("%s\n", fnd);
}
```

Εύρεση του **char** 'ch', στο string str1. Επιστροφή δείκτη **char** στην πρώτη εύρεση.

Αποτέλεσμα:

```
>  o you!
>
```



example_strchr.cpp

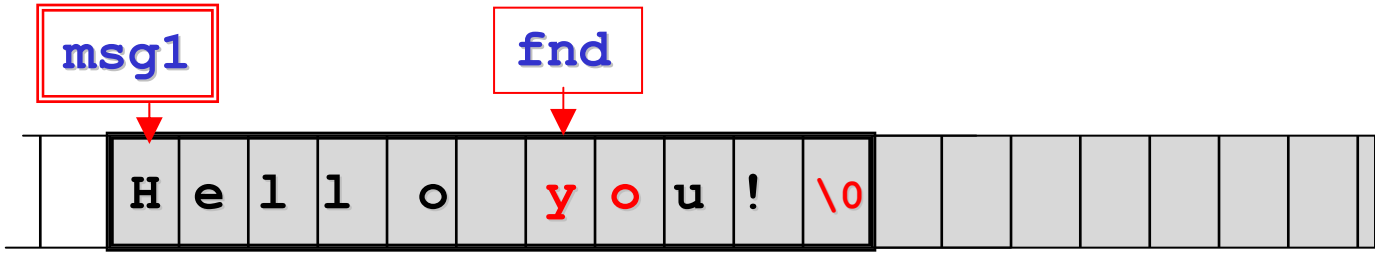
Εύρεση string: strstr(str1,str2)

```
#include <stdio.h>        /* για την printf()    */
#include <string.h>       /* για την strcpy()    */

main()
{
char msg1[81]={"Hello you!"};
char *fnd;                /* pointer to char */


    fnd = strstr(msg1,"yo");
    printf("%s\n", fnd);
}
```

Εύρεση του **string** str2 μέσα στο str1; Επιστροφή δείκτη **char** στην πρώτη εύρεση.

Αποτέλεσμα:

```
>  you!
>
```

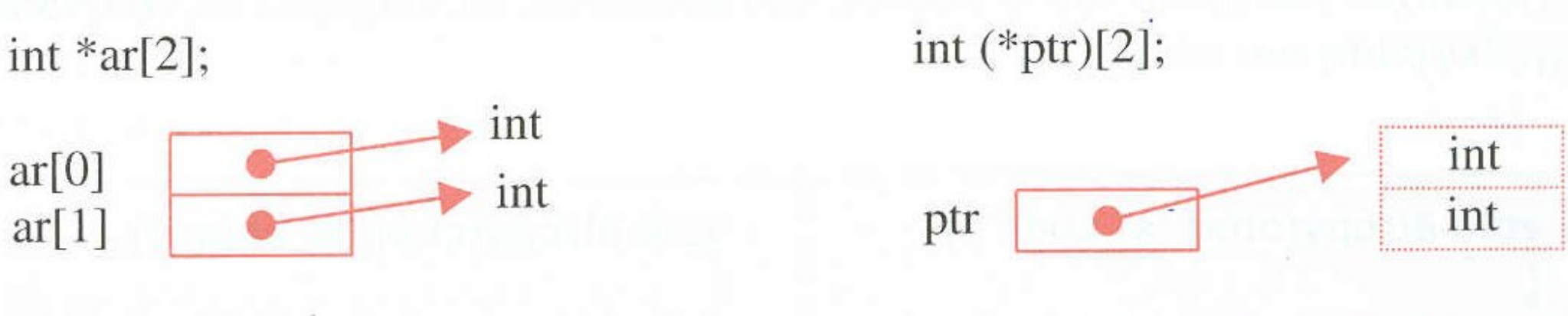


Παρατήρηση:

Σε μία έκφραση δήλωσης, ο τελεστής πίνακα έχει μεγαλύτερη προτεραιότητα από τον τελεστή *****. *Οι δύο παρακάτω δηλώσεις βασίζονται στο γεγονός αυτό:*

```
int *ar[2];
int (*ptr)[2];
```

Η πρώτη δήλωση ορίζει έναν πίνακα δύο δεικτών σε ακεραίους και στο χρόνο εκτέλεσης έχει ως αποτέλεσμα, όπως φαίνεται στο ακόλουθο σχήμα, τη δέσμευση δύο θέσεων μνήμης για μελλοντική αποθήκευση δεικτών σε ακεραίους. Αντίθετα, η δεύτερη δήλωση ορίζει ένα δείκτη σε πίνακα δύο ακεραίων, τον οποίο όμως δε δηλώνει και κατά συνέπεια δε δεσμεύει τον απαιτούμενο χώρο.



(α) Πίνακας 2 δεικτών σε ακέραιο.

(β) Δείκτης σε πίνακα 2 ακεραίων.

Διαχείριση μνήμης: οι τελεστές new-delete

Η C++ παρέχει τον τελεστή ***new*** για διαχείριση μνήμης. Ο συγκεκριμένος τελεστής αποκτά μνήμη από το λειτουργικό σύστημα και επιστρέφει ένα δείκτη για το σημείο αρχής του τμήματος:

```
include <stdio.h>
include <string.h>
main()
{
  char *str="Hello world";
  int len=strlen(str);
  char *ptr;
  ptr=new char[len+1]; //δέσμευση μνήμης για str + '\0'
  strcpy(ptr,str); //αντιγραφή του str στη νέα περιοχή μνήμης ptr
  printf( "ptr=%s",ptr );
  delete ptr; //αποδέσμευση της μνήμης του ptr
}
```

Αποτέλεσμα: > ptr=Hello world

Διαχείριση μνήμης: οι τελεστές new-delete

•Η παράσταση `ptr=new char[len+1];` επιστρέφει ένα δείκτη που δείχνει προς ένα τμήμα μνήμης αρκετό για να χωρέσει το ***str***, με ένα επιπλέον byte για το μηδενικό χαρακτήρα. Για το μέγεθος (***len+1***) θα πρέπει να χρησιμοποιηθούν **αγκύλες**. Η έλλειψη αγκυλών δε θα ανιχνευθεί από το μεταγλωττιστή αλλά θα οδηγήσει σε λάθη κατά την εκτέλεση του προγράμματος.

•Ο τελεστής ***new*** παίζει παρόμοιο ρόλο με την οικογένεια συναρτήσεων βιβλιοθήκης ***malloc***. Όμως, η νέα προσέγγιση είναι καλύτερη καθώς επιστρέφει ένα δείκτη για τον κατάλληλο τύπο δεδομένων, ενώ ο δείκτης της ***malloc*** πρέπει να προσαρμοσθεί στον κατάλληλο τύπο.

•Ο τελεστής ***delete*** απελευθερώνει τη δεσμευμένη μνήμη. Η διαγραφή της μνήμης δε διαγράφει το δείκτη που τη δείχνει (τον ***str*** στο προηγούμενο παράδειγμα) κι ούτε αλλάζει την τιμή της διεύθυνσης στο δείκτη. Ωστόσο, αυτή η διεύθυνση παύει να ισχύει. Η μνήμη στην οποία δείχνει μπορεί να συμπληρωθεί από κάτι άλλο εντελώς διαφορετικό. Θα πρέπει να προσεχθεί ώστε να μη χρησιμοποιούνται δείκτες που έχουν διαγραφεί με τον τελεστή ***delete***.

Σύνταξη του τελεστή new

```
char *ptr;
```

ίδιοι τύποι δεδομένων

αγκύλες

```
ptr=new char[len+1];
```

δείκτης

δεσμευμένη λέξη

τύπος δεδομένων των μεταβλητών

αριθμός των μεταβλητών τύπου **char**

Μνήμη που δεσμεύθηκε με τον τελεστή new

```
ptr=new char[len+1];
```