Διαδικαστικός Προγραμματισμός

Ενότητα 5: Πίνακες

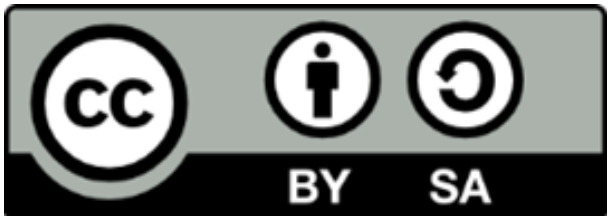
Καθηγήτρια Μαρία Σατρατζέμη

Τμήμα Εφαρμοσμένης Πληροφορικής

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοποί ενότητας

- Να εκτιμήσετε τη σημασία που έχουν για τον προγραμματισμό οι *πίνακες δεδομένων*.
- Να μπορείτε να δηλώνετε και να χειρίζεστε απλούς πίνακες.
- Να κατανοήσετε τον τρόπο με τον οποίο αποθηκεύονται στη μνήμη οι τιμές δεδομένων.
- Να αναγνωρίσετε σε τι διαφέρει η μεταβίβαση πινάκων ως παραμέτρων συναρτήσεων από τη μεταβίβαση απλών μεταβλητών.
- Να μάθετε πώς να κάνετε στατική ανάθεση αρχικών τιμών σε πίνακες.
- Να κατανοήσετε τη δομή των πολυδιάστατων πινάκων.

Σύνθετοι τύποι δεδομένων

- Μέχρι τώρα επικεντρώσαμε την προσοχή μας στη χρήση αλγορίθμων με ολοένα μεγαλύτερη πολυπλοκότητα για το χειρισμό απλών δεδομένων.
- Αυτούς τους απλούς τύπους δεδομένων που γνωρίσαμε στα προηγούμενα μαθήματα μπορούμε να τους χρησιμοποιήσουμε για να ορίσουμε πιο **σύνθετους τύπους δεδομένων**, προκειμένου να λύσουμε πιο πολύπλοκα προβλήματα.
- Όπως θα δούμε οι τύποι δεδομένων σχηματίζουν και αυτοί μια ιεραρχία και μας δίνουν τη δυνατότητα να συνδυάζουμε πολλές ανεξάρτητες τιμές δεδομένων με τέτοιον τρόπο ώστε να θεωρούνται μια ενοποιημένη συλλογή.
- Οι έννοιες δομή ελέγχου και **δομή δεδομένων** (data structure) σε συνδυασμό μεταξύ τους, αποτελούν τα θεμέλια του σύγχρονου προγραμματισμού.
- Το 1976, ο Niklaus Wirth, δημιουργός της γλώσσας προγραμματισμού Pascal, διατύπωσε την παραπάνω αρχή στον τίτλο ενός συγγράμματος προγραμματισμού με τη μορφή της παρακάτω εξίσωσης:

Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα

Εισαγωγή στους πίνακες (1)

- **Πίνακας** (array) είναι μια συλλογή μεμονωμένων τιμών δεδομένων, η οποία διαθέτει δύο χαρακτηριστικά:
  1. *Ο πίνακας είναι διατεταγμένος* – τα μεμονωμένα συστατικά στοιχεία ενός πίνακα μπορούμε να τα ξεχωρίσουμε ως πρώτο, δεύτερο, κ.ο.κ.
  2. *Ο πίνακας είναι ομοιογενής* – όλες οι τιμές που αποθηκεύονται σε έναν πίνακα είναι του ίδιου τύπου.

- Κάθε μία από τις τιμές ενός πίνακα ονομάζεται **στοιχείο** (element) αυτής της συνάρτησης.

Πίνακας 5 στοιχείων

Δήλωση πίνακα (1)

- Στη C, κάθε πίνακας έχει δύο βασικές ιδιότητες:
  - τον **τύπο στοιχείου** (element type), δηλαδή τον τύπο των τιμών οι οποίες είναι δυνατό να αποθηκευτούν στα στοιχεία του πίνακα
  - το **μέγεθος πίνακα** (array size), δηλαδή το πλήθος των στοιχείων που περιέχει ο πίνακας.
- Κάθε φορά που δημιουργείτε ένα νέο πίνακα στο πρόγραμμα σας, θα πρέπει να καθορίζετε τόσο τον τύπο των στοιχείων όσο και το μέγεθος του πίνακα:

***τύπος_στοιχείου όνομα_πινακα*[*μέγεθος*];**

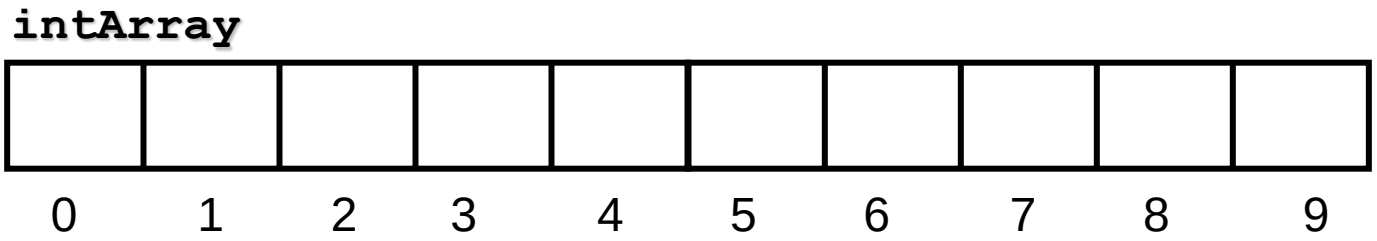
όπου*:*
- ***τύπος_στοιχείου*** είναι ο τύπος κάθε στοιχείου του πίνακα
- ***όνομα_πίνακα*** είναι το όνομα της μεταβλητής που δηλώνεται ως πίνακας
- ***μέγεθος*** είναι το πλήθος των στοιχείων που δεσμεύονται για τον πίνακα

Δήλωση πίνακα (2)

Παράδειγμα:

Πίνακας 10 θέσεων για ακεραίους (**int**) με όνομα **intArray**

int intArray[10];



- Κάθε στοιχείο του πίνακα προσδιορίζεται με μια αριθμητική τιμή που ονομάζεται **αριθμοδείκτης** (index) του στοιχείου.
- Στη C, οι αριθμοδείκτες σε έναν πίνακα ξεκινούν πάντα από το 0 και φτάνουν μέχρι το μέγεθος του πίνακα μείον ένα.
- Στις περισσότερες περιπτώσεις, θα πρέπει να καθορίζετε το μέγεθος του πίνακα χρησιμοποιώντας μια σταθερά, προκειμένου να διευκολύνετε τους προγραμματιστές που θα κληθούν να συντηρήσουν το πρόγραμμά σας στο μέλλον.

```
#define NElements 10
int intArray[NElements];
```

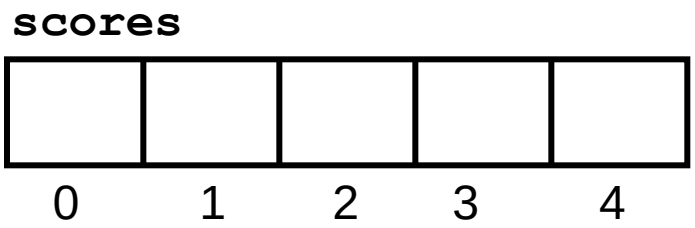
Δήλωση πίνακα (3)

Πρόβλημα: να γράψετε ένα πρόγραμμα, στο οποίο θα διαβάζετε τη βαθμολογία που δίνουν 5 κριτές στην προσπάθεια ενός αθλητή και θα υπολογίζετε και εμφανίζετε τον μέσο όρο βαθμολογίας.

Σε αυτό το πρόγραμμα μπορούμε να χρησιμοποιήσουμε ένα πίνακα 5 θέσεων για την αποθήκευση των βαθμολογιών:

```
#define NJudges 5

double scores[NJudges];
```



Επιλογή στοιχείων πίνακα

- Για να αναφερθούμε σε ένα συγκεκριμένο στοιχείο ενός πίνακα, θα πρέπει να προσδιορίσουμε τόσο το όνομα του πίνακα όσο και τον αριθμοδείκτη που αντιστοιχεί στη θέση αυτού του στοιχείου, διεργασία γνωστή ως **επιλογή** (selection).

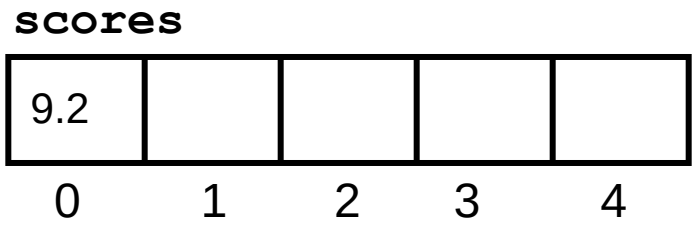
- Η επιλογή ενός στοιχείου από κάποιον πίνακα γίνεται με μια **παράσταση επιλογής** (selection expression) που έχει την ακόλουθη μορφή:

***όνομα_πίνακα*[*αριθμοδείκτης*]**

Π.χ. scores[0] = 9.2;

```
printf("%.2f", scores[0]);

for(i = 0; i < NJudges; i++){
    scores[i] = 0;
}
```



Επίλυση προβλήματος με πίνακα

```
                /*  Αρχείο: gymjudge.c
        Το πρόγραμμα βρίσκει το μέσο όρο από 5 βαθμολογίες κριτών. */
#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

#define NJudges 5                              /* Σταθερά – αριθμός κριτών */

main()
{
   double gymnasticScores[NJudges];    /*Δήλωση πίνακα βαθμολογιών κριτών */
   double total, average;
   int i;
       /* Διάβασμα των βαθμολογιών στον πίνακα NJudges */
   printf("Please enter a score for each judge.\n");
   for (i = 0; i < NJudges; i++) {
      printf("Score for judge #%d:  ", i);
      gymnasticScores[i] = GetReal();
   }
       /* Υπολογισμός του μέσου όρου των τιμών του πίνακα NJudges */
        total = 0;
   for (i = 0; i < NJudges; i++) {
      total += gymnasticScores[i];
   }
   average = total / NJudges;
   printf("The average score is %.2f\n", average);
}
```

Αλλαγή του διαστήματος των αριθμοδεικτών

```
#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

#define NJudges 5                          /* Σταθερά – αριθμός κριτών */

main()
{
   double gymnasticScores[NJudges+1];   /*Δήλωση πίνακα βαθμολογιών κριτών */
   double total, average;
   int i;
       /* Διάβασμα των βαθμολογιών στον πίνακα NJudges */
   printf("Please enter a score for each judge.\n");
   for (i = 1; i <= NJudges; i++) {
      printf("Score for judge #%d:  ", i);
      gymnasticScores[i] = GetReal();
   }
        /* Υπολογισμός του μέσου όρου των τιμών του πίνακα NJudges */
        total = 0;
   for (i = 1; i <= NJudges; i++) {
      total += gymnasticScores[i];
   }
   average = total / NJudges;
   printf("The average score is %.2f\n", average);
}
```

Εσωτερική αναπαράσταση δεδομένων

- Όλες οι τιμές δεδομένων αποθηκεύονται στο εσωτερικό του υπολογιστή με τη μορφή βασικών μονάδων πληροφοριών, οι οποίες ονομάζονται **bit** (binary digit).
- Σε κάθε bit αποθηκεύετε η «μικροσκοπική» πληροφορία **0 ή 1**.
- Για την αποθήκευση πιο «παραδοσιακών» τύπων πληροφοριών (όπως οι αριθμοί και οι χαρακτήρες) χρησιμοποιούνται μεγαλύτερες ομάδες από bit που αντιμετωπίζονται ως ενιαίες μονάδες αποθήκευσης και ονομάζονται byte.
- Ένα **byte** ισούται με 8 bit και αρκεί για την αποθήκευση ενός χαρακτήρα.
- Στα περισσότερα μηχανήματα, τα byte συγκεντρώνονται σε μεγαλύτερες δομές που ονομάζονται **λέξεις** (words) και ορίζονται, συνήθως, έτσι ώστε να έχουν το μέγεθος που απαιτείται για την αποθήκευση μιας ακέραιης τιμής.
- Η μνήμη των υπολογιστών μετριέται σε πολλαπλάσια του byte, όπως

  1 KB = 2^{10} byte, 1 MB = 2^{20} byte,…

Διευθύνσεις μνήμης (1)

- Στο εσωτερικό του συστήματος μνήμης, κάθε byte προσδιορίζεται με μια αριθμητική **διεύθυνση** (address).

- Σε κάθε byte της μνήμης μπορεί να αποθηκευτεί 1 χαρακτήρας πληροφορίας.

- Για παράδειγμα, αν δηλώσουμε τη μεταβλητή **ch**

  char ch;

  ο μεταγλωττιστής θα δεσμεύσει γι' αυτή 1 byte στη μνήμη, έστω αυτό που βρίσκεται στη διεύθυνση 1000.

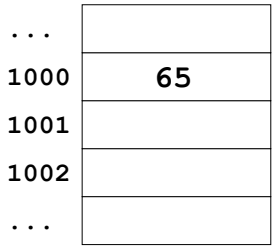
  Όταν εκτελεστεί η εντολή

  ch = 'A';

  η εσωτερική αναπαράσταση του χαρακτήρα 'A' (δηλ.

  ο κωδικός ASCII του χαρακτήρα που είναι 65)

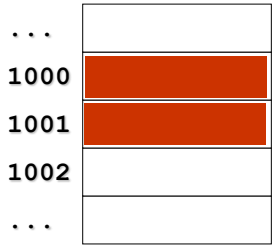
  θα αποθηκευτεί στη μνήμη 1000.



Διευθύνσεις μνήμης (2)

- Οι τιμές που είναι μεγαλύτερες από ένα χαρακτήρα ή το μέγεθος του απαιτεί περισσότερα από 1 byte αποθηκεύονται σε συνεχόμενες θέσεις μνήμης.

- Για παράδειγμα, αν σε ένα υπολογιστή οι ακέραιοι καταλαμβάνουν 2 **byte**, για την αποθήκευση ενός ακεραίου θα δεσμευτούν 2 συνεχόμενες θέσεις μνήμης όπως φαίνεται στο σχήμα:



- Οι τιμές τύπου **double** απαιτούν συνήθως 8 byte, οπότε για μια μεταβλητή τύπου **double** θα δεσμεύονταν 8 συνεχόμενες θέσεις.

Ο τελεστής sizeof

- Εκτός από τον τύπο δεδομένων **char**, ο οποίος έχει εξ ορισμού μήκος ενός byte σε όλα τα υπολογιστικά συστήματα, το πλήθος των byte που απαιτούνται για την αποθήκευση μιας τιμής ενός συγκεκριμένου τύπου ενδέχεται να διαφέρει από το ένα μηχάνημα στο άλλο.

- Στη C, για να μάθετε την ποσότητα μνήμης που θα δεσμευτεί για μια συγκεκριμένη μεταβλητή μπορείτε να χρησιμοποιήσετε τον τελεστή **sizeof** με τελεστέο
  - το όνομα ενός τύπου σε παρενθέσεις:

    sizeof(int)

    οπότε θα επιστρέψει το πλήθος των byte που απαιτούνται για την αποθήκευση μιας τιμής τύπου **int**
  - μια παράσταση – π.χ. μια μεταβλητή

    sizeof x

    οπότε θα επιστρέψει το πλήθος των byte που απαιτούνται για την αποθήκευση της μεταβλητής **x**

Κατανομή μνήμης σε μεταβλητές (1)

- Κάθε φορά που δηλώνετε μια μεταβλητή σε ένα πρόγραμμα, ο μεταγλωττιστής δεσμεύει χώρο μνήμης για την αποθήκευση της τιμής αυτής της μεταβλητής, διεργασία γνωστή ως **κατανομή** (allocation).

- Για τις **καθολικές μεταβλητές** (global variables), δηλαδή τις μεταβλητές που δηλώνονται στην αρχή του αρχείου πηγαίου κώδικα (έξω από κάθε συνάρτηση) και έχουν εμβέλεια σε όλο το αρχείο, η κατανομή μνήμης γίνεται όταν ξεκινά το πρόγραμμα. Οι μεταβλητές αυτές παραμένουν στις ίδιες διευθύνσεις μνήμης μέχρι να τερματιστεί το πρόγραμμα.

- Για τις **τοπικές μεταβλητές** (local variables) κατανέμεται μνήμη μόνο όταν καλείται η συνάρτηση στην οποία δηλώνονται, και όταν η συνάρτηση ολοκληρώσει την εκτέλεση της η μνήμη αποδεσμεύεται.

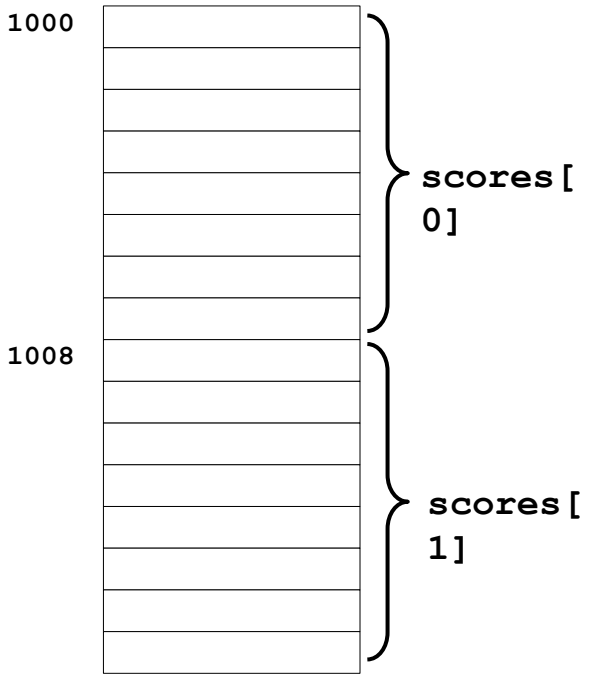
Κατανομή μνήμης σε μεταβλητές (2)

Για παράδειγμα, αν υποθέσουμε ότι η σταθερά **NJudges** έχει οριστεί ίση με 5 και ότι κάθε τιμή τύπου **double** απαιτεί 8 byte τότε για τον πίνακα **scores**

double scores[NJudges];

Θα απαιτηθούν

8 byte/στοιχείο × 5 στοιχεία = 40 byte



Όταν ο μεταγλωττιστής της C συναντά μια παράσταση επιλογής, όπως η **scores[i]** υπολογίζει την κατάλληλη διεύθυνση της μνήμης πολλαπλασιάζοντας τον αριθμοδείκτη με το μέγεθος κάθε στοιχείου και προσθέτοντας το γινόμενο στην αρχική διεύθυνση του πίνακα.

Π.χ. για το στοιχείο **scores[1]**

η διεύθυνση είναι: 1 × 8 + 1000 = 1008

Η αρχική διεύθυνση του πίνακα ονομάζεται **διεύθυνση βάσης** (base address), ενώ η απαραίτητη προσαρμογή για τον εντοπισμό ενός στοιχείου (1×8=8) **σχετική απόσταση** (offset).

Αναφορές σε στοιχεία εκτός ορίων πίνακα

- Όταν χρησιμοποιείτε πίνακες στα προγράμματά σας πρέπει να εξασφαλίζετε ότι οι τιμές των αριθμοδεικτών που χρησιμοποιείτε για να επιλέγετε στοιχεία από τους πίνακες θα παραμένουν εντός των ορίων των πινάκων.

- Στους περισσότερους υπολογιστές, η αναφορά σε στοιχεία που βρίσκονται εκτός των ορίων ενός πίνακα δεν εντοπίζεται ως λάθος, αλλά είναι σχεδόν βέβαιο ότι θα οδηγήσει σε απρόβλεπτα αποτελέσματα.

- Για παράδειγμα, για τον πίνακα **scores** που έχει 5 στοιχεία – με αριθμοδείκτες 0, 1, 2, 3, 4 – η αναφορά στο στοιχείο

  scores[5]

  θα έχει –πιθανόν- ως αποτέλεσμα την επιλογή της τιμής που είναι αποθηκευμένη στη διεύθυνση

  $5 \times 8 + 1000 = 1040$

  ενώ το τελευταίο στοιχείο του πίνακα είναι αποθηκευμένο στις θέσεις μνήμης 1032 έως 1039.

Μεταβίβαση πινάκων ως παραμέτρων (1)

```
#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

#define NJudges 5

void Read_Array(double array[]);
void Calculate_Average(double array[]);

main()
{
   double gymnasticScores[NJudges];

        Read_Array(gymnasticScores);
        Calculate_Average(gymnasticScores);
}
```

```
Please enter a score for each judge.
Score for judge #0:  9.2
Score for judge #1:  9.9
Score for judge #2:  9.7
Score for judge #3:  9.0
Score for judge #4:  9.5
The average score is 9.46
```

```
void Read_Array(double array[])
{
       int i;

       printf("Please enter a score for each judge.\n");
   for (i = 0; i < NJudges; i++) {
      printf("Score for judge #%d:  ", i);
      array[i] = GetReal();
   }
}
```

```
void Calculate_Average(double array[])
{
       double total, average;
       int i;

       total = 0;
  for (i = 0; i < NJudges; i++) {
     total += array[i];
  }
  average = total / NJudges;
  printf("The average score is %.2f\n", average);
}
```

Μεταβίβαση πινάκων ως παραμέτρων (2)

- Αν καλέσετε μια συνάρτηση που δέχεται ως τυπική παράμετρο μια μεταβλητή βασικού τύπου, όπως **int**, **long** και **double**, τότε η συνάρτηση λαμβάνει ένα αντίγραφο της τιμής-εισόδου. Αυτό σημαίνει ότι η συνάρτηση δεν λαμβάνει την πρωτότυπη τιμή και συνεπώς, ακόμα και αν αλλάξει η τιμή μέσα στη συνάρτηση, δεν μπορεί η συνάρτηση να αλλάξει την τιμή της μεταβλητής-ορίσματος.

- Στην περίπτωση συνάρτησης με τυπική παράμετρο ένα πίνακα δεν ισχύει ότι ισχύει για τις απλές μεταβλητές.

- Αν καλέσετε μια συνάρτηση που δέχεται ως τυπική παράμετρο έναν πίνακα, ο χώρος αποθήκευσης του πίνακα ο οποίος θα χρησιμοποιηθεί από την παράμετρο είναι ίδιος με αυτόν που χρησιμοποιείται για το πραγματικό όρισμα.

- Συνεπώς, **η αλλαγή της τιμής ενός στοιχείου του πίνακα της παραμέτρου θα έχει ως αποτέλεσμα την αλλαγή της τιμής του αντίστοιχου στοιχείου στον πίνακα του ορίσματος**.

Μεταβίβαση πινάκων ως παραμέτρων (3)

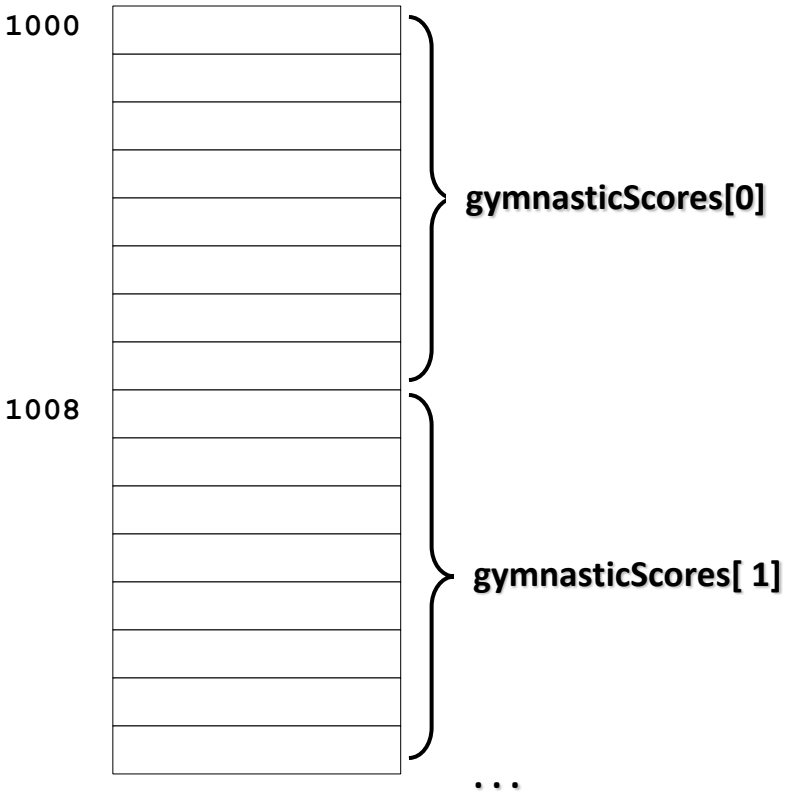
- Στη C, μια συνάρτηση που δέχεται ως όρισμα έναν πίνακα, επεξεργάζεται απευθείας τις τιμές που είναι αποθηκευμένες στις διευθύνσεις της μνήμης που αντιστοιχούν στον πίνακα.

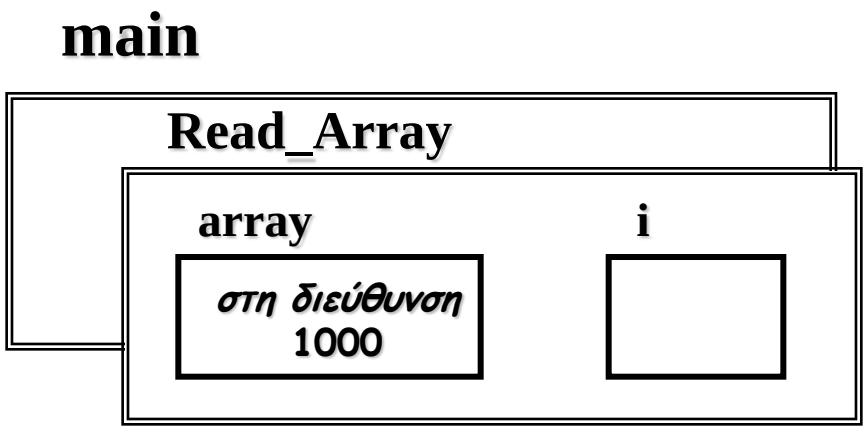
- Κάθε φορά που μεταβιβάζεται ένας πίνακας σε μια συνάρτηση, στο τοπικό πλαίσιο καταγράφεται μόνο η διεύθυνση βάσης του πίνακα.

- Αν επιλέξετε ένα στοιχείο του τοπικού πίνακα, ο οποίος έχει δηλωθεί στο εσωτερικό της συνάρτησης, η άθροιση της σχετικής απόστασης στη διεύθυνση βάσης θα έχει αποτέλεσμα την παραγωγή μιας διεύθυνσης η οποία καταλαμβάνεται από τον πίνακα-όρισμα με τον οποίο καλείται η συνάρτηση.

- Το τελικό αποτέλεσμα είναι ότι η τυπική παράμετρος πίνακα που έχει δηλωθεί στην κεφαλίδα μιας συνάρτησης καταλήγει να αποτελεί συνώνυμο του πίνακα ο οποίος μεταβιβάζεται ως όρισμα, και όχι αντίγραφό του.

Μεταβίβαση πινάκων ως παραμέτρων (4)



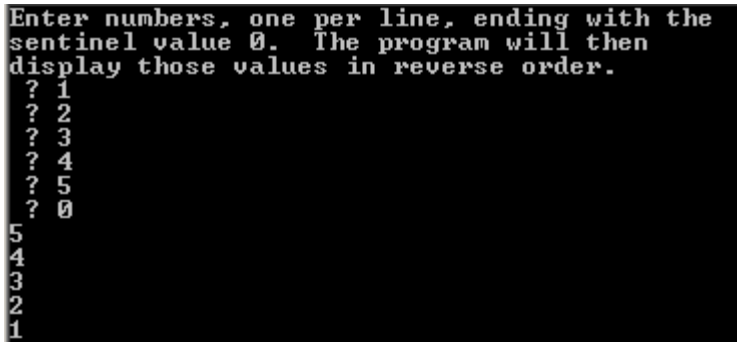


Παράδειγμα: αντιμετάθεση στοιχείων πίνακα (1)

Πρόβλημα: να γράψετε ένα πρόγραμμα που να εκτελεί τα παρακάτω βήματα:

1. Να διαβάζει μια λίστα ακεραίων μέχρι να καταχωρίσει ο χρήστης την τιμή-φρουρό 0.
2. Να αντιστρέφει τα στοιχεία αυτής της λίστας.
3. Να εμφανίζει τη λίστα με αντίστροφη σειρά.

Στη συνέχεια, παρουσιάζεται ένα δείγμα εκτέλεσης του προγράμματος:



Παράδειγμα: αντιμετάθεση στοιχείων πίνακα (2)

```
#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

#define MaxElements 250
#define Sentinel     0

static int GetIntegerArray(int array[], int max, int sentinel);
static void PrintIntegerArray(int array[], int n);
static void ReverseIntegerArray(int array[], int n);
static void SwapIntegerElements(int array[], int p1, int p2);
static void GiveInstructions(void);

main()
{
   int list[MaxElements], n;

   GiveInstructions();
   n = GetIntegerArray(list, MaxElements, Sentinel);
   ReverseIntegerArray(list, n);
   PrintIntegerArray(list, n);
}
```

Διάσταση πίνακα, ορίζεται στη δήλωση του πίνακα

τρέχον μέγεθος πίνακα

Παράδειγμα: αντιμετάθεση στοιχείων πίνακα (3)

```
static int GetIntegerArray(int array[], int max, int sentinel)
{
   int n, value;

   n = 0;
   while (TRUE) {
      printf(" ? ");
      value = GetInteger();
      if (value == sentinel) break;
      if (n == max) Error("Too many input items for array");
      array[n] = value;
      n++;
   }
   return (n);
}

static void PrintIntegerArray(int array[], int n)
{
   int i;

   for (i = 0; i < n; i++) {
      printf("%d\n", array[i]);
   }
}
```

Παράδειγμα: αντιμετάθεση στοιχείων πίνακα (4)

```
static void ReverseIntegerArray(int array[], int n)
{
   int i;

   for (i = 0; i < n / 2; i++) {
      SwapIntegerElements(array, i, n - i - 1);
   }
}

static void SwapIntegerElements(int array[], int p1, int p2)
{
   int tmp;

   tmp = array[p1];
   array[p1] = array[p2];
   array[p2] = tmp;
}

static void GiveInstructions(void)
{
   printf("Enter numbers, one per line, ending with the\n");
   printf("sentinel value %d.  The program will then\n", Sentinel);
   printf("display those values in reverse order.\n");
}
```

Λειτουργικές μονάδες

- Όταν ένα πρόγραμμα περιλαμβάνει πολλές συναρτήσεις, για παράδειγμα 50 ή και περισσότερες, είναι δύσκολο να διαχειριστεί και να κατανοηθεί.

- Σε μια τέτοια περίπτωση, η καλύτερη στρατηγική είναι να διαιρεθεί το πρόγραμμα σε μικρότερα πηγαία αρχεία, καθένα από τα οποία θα περιλαμβάνει ένα σύνολο σχετικών μεταξύ τους συναρτήσεων.

- Αυτά τα μικρότερα αρχεία, καθένα από τα οποία αποτελεί ένα μέρος του όλου προγράμματος και περιλαμβάνει σχετικές μεταξύ τους συναρτήσεις, ονομάζονται **λειτουργικές μονάδες** (modules).

- Η λειτουργική μονάδα που περιέχει τη συνάρτηση **main** ονομάζεται κύρια λειτουργική μονάδα (main module) και βρίσκεται στο υψηλότερο επίπεδο της ιεραρχίας.

- Κάθε μια από τις βοηθητικές λειτουργικές μονάδες ενεργεί ως βιβλιοθήκη για την κύρια λειτουργική μονάδα. Για κάθε τέτοια λειτουργική μονάδα θα πρέπει να υπάρχει και το αντίστοιχο αρχείο διασύνδεσης.

Εμβέλεια μεταβλητών

- Το τμήμα ενός προγράμματος στο οποίο μπορεί να χρησιμοποιηθεί μια μεταβλητή ονομάζεται **εμβέλεια** (scope) της μεταβλητής. Η εμβέλεια μιας μεταβλητής διαφέρει ανάλογα με το αν αυτή έχει δηλωθεί ως τοπική, καθολική ή στατική.
- **Καθολικές μεταβλητές** (global variables): μεταβλητές που δηλώνονται έξω από οποιονδήποτε ορισμό συνάρτησης και έχουν εμβέλεια μέσα σε όλο το αρχείο πηγαίου κώδικα (όλες τις συναρτήσεις συμπεριλαμβανομένης και της **main**). Η χρήση τους πρέπει να αποφεύγεται.
- **Τοπικές μεταβλητές** (local variables): μεταβλητές που δηλώνονται στο εσωτερικό μιας συνάρτησης και η εμβέλεια τους περιορίζεται μόνο στην ίδια τη συνάρτηση. Όταν γίνεται επιστροφή από τη συνάρτηση, όλες οι τοπικές μεταβλητές «εξαφανίζονται».
- **Στατικές μεταβλητές** (static variables): μεταβλητές που δηλώνονται με τη λέξη **static** έξω από κάθε συνάρτηση και έχουν εμβέλεια μέσα σε μια λειτουργική μονάδα:

```
static int x;
```

Ιδιωτικές Συναρτήσεις

- Η λέξη **static** μπορεί να χρησιμοποιηθεί για να δηλωθεί ότι μια συνάρτηση είναι ιδιωτική για μια συγκεκριμένη λειτουργική μονάδα.
- Σε πολλές περιπτώσεις όταν ορίζεται μια λειτουργική μονάδα ενδέχεται να περιλαμβάνει συναρτήσεις που καλούνται μόνο μέσα από τη λειτουργική μονάδα στην οποία ορίζονται.
- Για να υποδείξετε ότι μια συγκεκριμένη συνάρτηση περιορίζεται σε μια συγκεκριμένη λειτουργική μονάδα μπορείτε να χρησιμοποιήσετε τη λέξη κλειδί **static** στην αρχή τόσο του πρωτοτύπου της συνάρτησης όσο και της υλοποίησης της. Με αυτόν τον τρόπο είναι αδύνατη η κλήση αυτής της συνάρτησης από άλλα προγράμματα (πελάτες).
- Η δήλωση στατικών συναρτήσεων έχει και το πλεονέκτημα ότι σε μεγάλου μεγέθους λογισμικό, που αναπτύσσεται από πολλούς προγραμματιστές, αν μια συνάρτηση (ή μια καθολική μεταβλητή) δηλωθεί ως static μπορεί να χρησιμοποιηθεί το ίδιο όνομα και σ’ άλλες λειτουργικές μονάδες.
- Οι σχεδιαστές ανεξάρτητων λειτουργικών ομάδων ή 1) επικοινωνούν για να μη χρησιμοποιήσουν τα ίδια ονόματα ή 2) τα ονόματα δηλώνονται ως static .

Στατική ανάθεση αρχικών τιμών σε πίνακες

- Οι μεταβλητές πίνακα είναι δυνατό να δηλωθούν είτε ως τοπικές είτε ως καθολικές, όπως και οποιαδήποτε άλλη μεταβλητή.
- Η δήλωση ενός πίνακα, όπως και κάθε άλλης μεταβλητής, ως καθολική πρέπει να αποφεύγεται.
- Αν χρειαστεί ένας πίνακας θα πρέπει να δηλωθεί ως στατική καθολική μεταβλητή.
- Όταν ένας πίνακας δηλωθεί ως στατική καθολική μεταβλητή, μπορεί ταυτόχρονα να του γίνει και ανάθεση αρχικών τιμών

static int digits[10] = {0,1,2,3,4,5,6,7,8,9};

ή

static int digits[] = {0,1,2,3,4,5,6,7,8,9};

αφού ο μεταγλωττιστής μετρά το πλήθος των αρχικών τιμών και δεσμεύει χώρο μνήμης τόσα στοιχεία όσα έχουμε αρχικοποιήσει.

Πολυδιάστατοι πίνακες (1)

- Στη C, τα στοιχεία ενός πίνακα μπορεί να είναι οποιουδήποτε τύπου.
- Συνεπώς, τα στοιχεία ενός πίνακα μπορεί να είναι και τα ίδια πίνακες.
- Οι πίνακες πινάκων ονομάζονται **πολυδιάστατοι πίνακες** (multidimensional arrays).
- Η συνηθέστερη μορφή πολυδιάστατου πίνακα είναι ο **διδιάστατος πίνακας**.
- Ένας διδιάστατος πίνακας χρησιμοποιείται συνήθως για την αναπαράσταση δεδομένων των οποίων οι μεμονωμένες καταχωρίσεις σχηματίζουν μια ορθογώνια δομή που είναι δυνατό να χωριστεί σε γραμμές και στήλες και ονομάζεται **μήτρα** (matrix).

Πολυδιάστατοι πίνακες (2)

- Έστω ότι σε κάποιο πρόγραμμα έπρεπε να αναπαραστήσετε στην οθόνη μια παρτίδα τρίλιζας.
- Εφόσον, η τρίλιζα παίζεται σε έναν πίνακα τριών γραμμών και τριών στηλών θα μπορούσαμε να ορίσουμε ένα διδιάστατο πίνακα χαρακτήρων, κάθε στοιχείο του οποίου μπορεί να έχει ως τιμή τον χαρακτήρα ‘ ’, ‘Χ’ ή ‘Ο’:

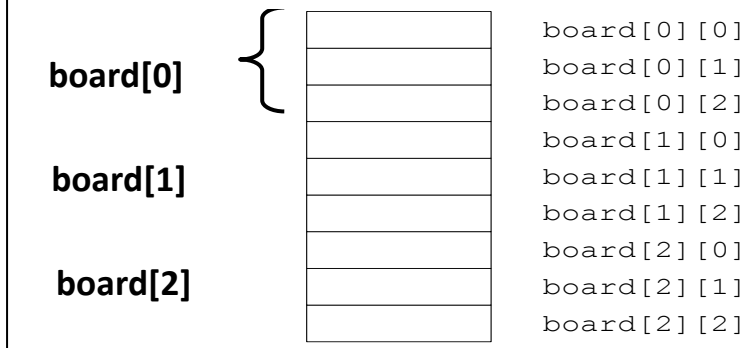
char board[3][3];

Γραμμές ⇒

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

⇑ Στήλες

Κατανομή μνήμης



Μεταβίβαση πολυδιάστατων πινάκων σε συναρτήσεις (1)

Η μεταβίβαση πολυδιάστατων πινάκων μεταξύ συναρτήσεων γίνεται όπως ακριβώς των μονοδιάστατων.

```
…
#define ROWS 8
#define COLUMNS 3
//δήλωση πρωτότυπου συνάρτησης
void ReadData(int matrix[ROWS][COLUMNS]);
main()
{
   int scores[ROWS][COLUMNS];

   ReadData(scores); //κλήση συνάρτησης
}
// υλοποίηση συνάρτησης
void ReadData(int matrix[ROWS][COLUMNS])
{
   int i,j;
   for (i=0;i<ROWS;i++)
                for (j=0;j<COLUMNS;j++)
                    matrix[i][j]=GetInteger();
}
```

Μεταβίβαση πολυδιάστατων πινάκων σε συναρτήσεις (2)

Όταν μια συνάρτηση δέχεται ως παράμετρο κάποιο πολυδιάστατο πίνακα η C απαιτεί τον καθορισμό του μεγέθους κάθε αριθμοδείκτη του πίνακα (διάσταση) εκτός του πρώτου. Κατά συνέπεια η κεφαλίδα της συνάρτησης ReadData θα μπορούσε να γραφεί .

```
...

//δήλωση πρωτότυπου συνάρτησης
void ReadData(int matrix[][COLUMNS]);

                                    Αντί

void ReadData(int matrix[ROWS][COLUMNS]);
```

Παράλειψη της δήλωσης της 1ης διάστασης (μέγεθος 1ου αριθμοδείκτη)

Σ' αυτή τη περίπτωση το πρόγραμμα θα λειτουργούσε με τον ίδιο τρόπο αφού η διεύθυνση βάσης του πίνακα και το μέγεθος του 2ου αριθμοδείκτη αρκούν για να προσδιορίσει τη διεύθυνση κάθε στοιχείου του πίνακα.

Επειδή η παράλειψη του πρώτου αριθμοδείκτη κάνει τη δήλωση κάπως ασύμμετρη γι' αυτό συνήθως συμπεριλαμβάνουμε στη δήλωση μιας παραμέτρου πολυδιάστατου πίνακα όλες τις διαστάσεις του πίνακα.

Ανάθεση αρχικών τιμών σε πίνακες (1)

```
static double identityMatrix[3][3]= {
        { 1.0, 0.0, 0.0 },
        { 0.0, 1.0, 0.0 },
        { 0.0, 0.0, 1.0 }
};
```

1.0	0.0	0.0
0.0	1.0	0.0
0.0	0.0	1.0

Η συγκεκριμένη μήτρα, η οποία εμφανίζεται πολύ συχνά σε μαθηματικές εφαρμογές, ονομάζεται **μοναδιαία μήτρα** (identity matrix).

Τέλος Ενότητας