ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Τμήμα Πληροφορικής και Τηλεπικοινωνιών • Εισαγωγή στον Προγραμματισμό

Περιεχόμενα

Κεφάλαιο 6: Δυναμική μνήμη, συμβολοσειρές και πολυδιάστατοι πίνακες

Δυναμική δέσμευση μνήμης

- Οι πίνακες είναι ένα πολύ χρήσιμο εργαλείο σε κάθε γλώσσα προγραμματισμού, άρα και στην C, για τη διαχείριση με ενιαίο τρόπο ενός συνόλου από ομοειδή δεδομένα. Το βασικό πρόβλημα, όμως, με τους πίνακες είναι ότι πρέπει να έχουμε ορίσει τη διάσταση τους μέσα στο πρόγραμμα, βάζοντας έτσι, κατά τη φάση της συγγραφής του προγράμματος, ένα όριο στο πλήθος των δεδομένων που μπορούμε να αποθηκεύσουμε στον πίνακα.
- Αν ορίσουμε τη διάσταση ενός πίνακα σχετικά μικρή, κάνουμε οικονομία στη μνήμη κατά την εκτέλεση του προγράμματος, αλλά αυτό έχει τότε περιορισμένες δυνατότητες. Αν ορίσουμε τη διάσταση πολύ μεγάλη, μπορεί το πρόγραμμά μας να χειριστεί και περισσότερα δεδομένα, αλλά γίνεται σπατάλη στη μνήμη όταν δεν χρειαζόμαστε να επεξεργαστούμε τόσο πολλά δεδομένα.

- Το ιδανικό θα ήταν το πρόγραμμά μας να μην δεσμεύει μνήμη εξ αρχής, αλλά να το κάνει αυτό δυναμικά, κατά τη φάση της εκτέλεσης, ανάλογα με τις ανάγκες του. Αυτό, ευτυχώς, είναι εφικτό.
- Η δυναμική δέσμευση μνήμης γίνεται σ' ένα χώρο που διαχειρίζεται το πρόγραμμα, ο οποίος λέγεται σωρός, και είναι διαφορετικός από το στατικό χώρο στον οποίο φυλάσσονται οι εξωτερικές μεταβλητής και τη στοίβα στην οποία φυλάσσονται οι αυτόματες μεταβλητές των συναρτήσεων.
- Στην C, η δυναμική δέσμευση μνήμης γίνεται (κυρίως) μέσω της συνάρτησης

  ```
  void *malloc(unsigned int size)
  ```

- Καλώντας την `malloc` με παράμετρο έναν ακέραιο `size`,[1] αυτή δεσμεύει στο σωρό `size` συνεχόμενα bytes και επιστρέφει στο όνομά της ένα δείκτη στο πρώτο απ' αυτά.
- Ο δείκτης που επιστρέφει η `malloc` δείχνει σε τύπο `void`, δηλαδή στον κενό τύπο, αφού δεν την αφορά τι τύπου δεδομένα θα φυλάξουμε στη μνήμη που δέσμευσε.
- Αν, για κάποιο λόγο, η `malloc` δεν μπορέσει να δεσμεύσει τη μνήμη που της ζητήθηκε, επιστρέφει στο όνομά της τον κενό δείκτη `NULL`. Πάντα, όταν καλούμε την `malloc`, πρέπει να ελέγχουμε αν μας επέστρεψε δείκτη διάφορο του `NULL` και μετά να προχωρήσουμε.
- Επειδή το πλήθος των bytes που χρησιμοποιούνται για τη φύλαξη δεδομένων κάποιου τύπου μπορεί να διαφέρει μεταξύ διαφορετικών μεταγλωττιστών, λειτουργικών συστημάτων και επεξεργαστών, για να είναι τα προγράμματά μας μεταφέρσιμα, το πλήθος των bytes που ζητάμε από την `malloc` να δεσμεύσει το δίνουμε μέσω του τελεστή `sizeof`.
- Η έκφραση `sizeof(`⟨τύπος⟩`)` έχει σαν τιμή το πλήθος των bytes που απαιτούνται στη συγκεκριμένη υλοποίηση της C για να φυλαχθούν δεδομένα που ο τύπος τους είναι ⟨τύπος⟩. Για παράδειγμα, με το παρακάτω τμήμα προγράμματος

  ```
  int n, *p;
  ..............                    /* Compute n */
  p = malloc(n * sizeof(int));
  if (p == NULL) {
     printf("Sorry, cannot allocate memory\n");
     return -1;
  }
  ..............  /* Handle data starting at p */
  ```

  δεσμεύουμε δυναμικά χώρο στο σωρό για να φυλαχθούν `n` ακέραιοι.
- Στην προ ANSI C εποχή, ο δείκτης που επέστρεφε η `malloc` (τύπου `void *`) έπρεπε πρώτα να προσαρμοσθεί στον κατάλληλο τύπο δείκτη και μετά να ανατεθεί σε μεταβλητή.

[1] Στον τυπικό ορισμό της `malloc`, ο τύπος του `size` είναι `size_t`, αλλά αυτό πρακτικά είναι `unsigned int`.

Δηλαδή την κλήση της `malloc` στο παραπάνω τμήμα προγράμματος θα την γράφαμε σαν:

```
p = (int *) malloc(n * sizeof(int));
```

Πλέον, κάτι τέτοιο δεν είναι απαραίτητο και, για την ακρίβεια, δεν συνίσταται.

- Ο τελεστής `sizeof` μπορεί να χρησιμοποιηθεί είτε σαν `sizeof` ⟨παράσταση⟩, είτε σαν `sizeof(`⟨παράσταση⟩`)`, οπότε αυτή η έκφραση έχει σαν τιμή το πλήθος των bytes που απαιτούνται για να φυλαχθεί η ⟨παράσταση⟩, π.χ. το `sizeof buf` είναι `20`, αν έχουμε ορίσει `char buf[20]`.
- ΠΡΟΣΟΧΗ! Όταν δεσμεύουμε δυναμικά μνήμη, είναι ευθύνη δική μας να την αποδεσμεύουμε όταν δεν την χρειαζόμαστε.
- Η αποδέσμευση μνήμης που έχει δεσμευθεί δυναμικά γίνεται με τη συνάρτηση

  ```
  void free(void *p)
  ```

- Η παράμετρος που περνάμε στη συνάρτηση `free` είναι ένας δείκτης που μας είχε επιστρέψει κάποια `malloc`.
- Είναι πολύ σοβαρό λάθος σ' ένα πρόγραμμα, να δεσμεύουμε δυναμικά μνήμη και να μην την αποδεσμεύουμε όταν δεν την χρειαζόμαστε.
- Εκτός από την `malloc`, υπάρχουν και δύο άλλες συναρτήσεις για δυναμική δέσμευση μνήμης (η `realloc` και η `calloc`). Είναι πιο σπάνια χρησιμοποιούμενες από την `malloc`. Μέσω της εντολής `man`, μπορεί να μάθει κανείς περισσότερες πληροφορίες γι' αυτές.
- Όταν σ' ένα πρόγραμμα χρησιμοποιούμε συναρτήσεις για δυναμική δέσμευση και αποδέσμευση μνήμης, πρέπει να έχουμε συμπεριλάβει στην αρχή το αρχείο επικεφαλίδας `stdlib.h`. Δηλαδή:

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

- Σε σχέση με τα προβλήματα που δημιουργούνται όταν ορίζουμε με στατικό τρόπο πίνακες για να διαχειριστούμε ένα σύνολο από δεδομένα, αντί να κάνουμε δυναμική δέσμευση μνήμης, θα πρέπει να αναφέρουμε ότι στην C υπάρχει και η δυνατότητα να ορίσουμε πίνακες με μεταβλητή διάσταση, για παράδειγμα:

  ```
  int array[n];
  ```

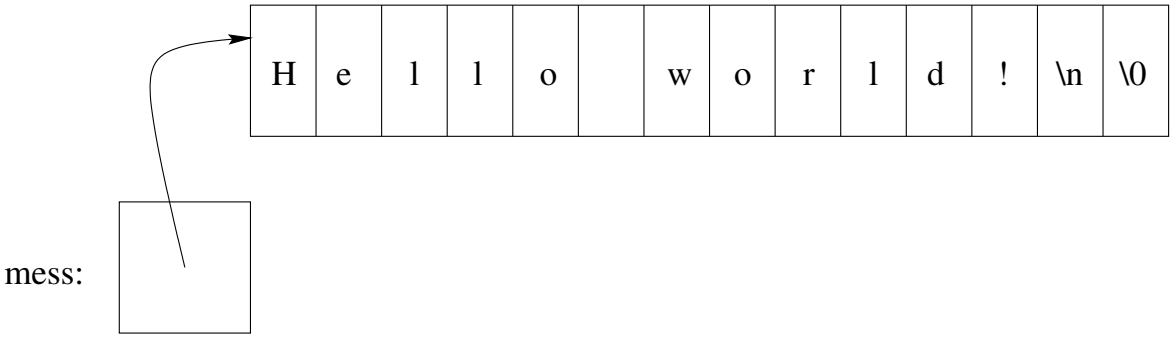
  Φυσικά, πριν από τη δήλωση αυτή, στην ακέραια μεταβλητή `n` πρέπει να έχουμε δώσει τιμή.
- Ο βασικός περιορισμός που υπάρχει στη δήλωση πινάκων με διάσταση που δεν είναι γνωστή κατά τη φάση συγγραφής του προγράμματος, είναι ότι μπορεί να γίνει για πίνακες που είναι αυτόματοι, δηλαδή τοπικοί μέσα σε συναρτήσεις. Δεν μπορεί να γίνει για πίνακες που θέλουμε να τους ορίσουμε στον στατικό χώρο, εξωτερικά των συναρτήσεων.

- Σε κάθε περίπτωση, είναι καλύτερη προγραμματιστική τακτική, όταν θέλουμε να δεσμεύσουμε δυναμικά μνήμη, αυτό να γίνεται μέσω `malloc` και δεικτών, παρά με πίνακες που η διάστασή τους είναι παραμετρική.
- Επίσης, να επισημανθεί ότι η δυνατότητα για πίνακες με παραμετρική διάσταση προστέθηκε στο στάνταρντ της C το 1999, οπότε παλιότεροι μεταγλωττιστές δεν το δέχονται.

Συμβολοσειρές

- Μία συμβολοσειρά ή αλφαριθμητικό είναι ένας πίνακας από χαρακτήρες (δεδομένα τύπου `char`). Ο χαρακτήρας `'\0'`, σαν στοιχείο του πίνακα, δείχνει το τέλος της συμβολοσειράς και είναι απολύτως απαραίτητος για να μπορούν να λειτουργήσουν οι συναρτήσεις βιβλιοθήκης για το χειρισμό συμβολοσειρών.
- Λόγω της άμεσης σχέσης μεταξύ πινάκων και δεικτών, σε μία συμβολοσειρά μπορούμε να αναφερθούμε και με ένα δείκτη (τύπου `char *`) που δείχνει στον πρώτο χαρακτήρα της συμβολοσειράς.
- Μέσα σ' ένα πρόγραμμα, μπορούμε μ' ένα σύντομο τρόπο να αναφερθούμε σε μία συμβολοσειρά για την οποία γνωρίζουμε τους χαρακτήρες που περιέχει, περικλείοντας αυτούς τους χαρακτήρες μέσα σε `"` (χωρίς το `'\0'`). Παράδειγμα:

```
char *mess = "Hello world!\n";
```



- Οι συμβολοσειρές που περιέχονται μέσα σ' ένα πρόγραμμα φυλάσσονται στον ίδιο χώρο με το πρόγραμμα και, γι' αυτόν το λόγο, δεν είναι δυνατόν να μεταβληθούν. Για τη διαχείρισή τους, ο μεταγλωττιστής αντιστοιχεί σε κάθε τέτοια συμβολοσειρά τη διεύθυνση του πρώτου χαρακτήρα της.
- Εκτός από την αρχικοποίηση ενός δείκτη σε `char` με μία συμβολοσειρά, όπως κάναμε με την εντολή

  ```
  char *mess = "Hello world!\n";
  ```

  μπορούμε να αρχικοποιήσουμε και ένα πίνακα χαρακτήρων με μία συμβολοσειρά, ως εξής:

  ```
  char arrmess[] = "How are you?";
  ```

Δεν υπάρχει λόγος να ορίσουμε διάσταση για τον πίνακα `arrmess`, γιατί αυτή υπολογίζεται αυτόματα από το πλήθος των χαρακτήρων της συμβολοσειράς (συν έναν, λόγω του `'\0'`).

- Μία ουσιώδης διαφορά μεταξύ των παραπάνω εντολών, είναι ότι η μεταβλητή `mess` είναι δείκτης που αρχικοποιήθηκε με τη συμβολοσειρά που θέλουμε, στη συνέχεια, όμως, μπορεί να αλλάξει τιμή, αν χρειάζεται, ενώ το όνομα του πίνακα `arrmess`, που επίσης αρχικοποιήθηκε με μία συμβολοσειρά, λειτουργεί, φυσικά, και ως δείκτης, μόνο που δεν επιτρέπεται να μεταβληθεί.
- Η δεύτερη ουσιώδης διαφορά είναι ότι δεν μπορούμε να αλλάξουμε το περιεχόμενο, δηλαδή τους χαρακτήρες, της συμβολοσειράς `"Hello world!\n"` που δείχνει αρχικά ο δείκτης `mess`, παρότι μπορούμε να αλλάξουμε την τιμή του ίδιου του δείκτη, ενώ μπορούμε να αλλάξουμε τους χαρακτήρες της συμβολοσειράς `"How are you?"`, παρότι δεν μπορούμε να αλλάξουμε το `arrmess`.
- Οι συμβολοσειρές είναι ένα πολύ χρήσιμο εργαλείο για τον προγραμματισμό στην C, γι' αυτό η πρότυπη βιβλιοθήκη της γλώσσας παρέχει μία σειρά από συναρτήσεις για τη διαχείρισή τους.
- Όταν χρησιμοποιούμε σ' ένα πρόγραμμα συναρτήσεις για διαχείριση συμβολοσειρών, πρέπει να έχουμε συμπεριλάβει στην αρχή το αρχείο επικεφαλίδας `string.h`. Δηλαδή:

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

- Μερικές από τις συναρτήσεις αυτές είναι:

  ```
  unsigned int strlen(const char *s)
  char *strcpy(char *s1, const char *s2)
  int strcmp(const char *s1, const char *s2)
  char *strcat(char *s1, const char *s2)
  ```

- Η `strlen` επιστρέφει το μήκος της συμβολοσειράς `s` (χωρίς το τελικό `'\0'`). [2]
- Η `strcpy` αντιγράφει τη συμβολοσειρά `s2`, μέχρι και το τελικό `'\0'`, στη συμβολοσειρά `s1`, την οποία επιστρέφει και στο όνομά της.
- Η `strcmp` συγκρίνει τις συμβολοσειρές `s1` και `s2` byte προς byte και επιστρέφει έναν ακέραιο θετικό, μηδέν ή αρνητικό, ανάλογα αν η συμβολοσειρά `s1` ακολουθεί (αλφαβητικά), ταυτίζεται ή προηγείται της συμβολοσειράς `s2`, αντίστοιχα. Η σύγκριση γίνεται με βάση τους ASCII κωδικούς των χαρακτήρων των συμβολοσειρών.
- Η `strcat` προσαρτά ένα αντίγραφο της συμβολοσειράς `s2` στο τέλος της `s1`, διαγράφοντας πρώτα το `'\0'` της `s1`, και επιστρέφει το αποτέλεσμα και στο όνομά της.

[2] Επισήμως ο τύπος επιστροφής της συνάρτησης είναι `size_t`, αλλά αυτός, πρακτικά, είναι `unsigned int`.

- Κάποιες από τις τυπικές παραμέτρους των συναρτήσεων για διαχείριση συμβολοσειρών έχουν δηλωθεί σαν `const` για να επισημανθεί ότι οι αντίστοιχες συμβολοσειρές δεν θα μεταβληθούν από τις συναρτήσεις.
- Οι συναρτήσεις που δημιουργούν νέες συμβολοσειρές (π.χ. `strcpy`, `strcat`) δεν αναλαμβάνουν και τη δέσμευση μνήμης για τη φύλαξή τους. Είναι ευθύνη της καλούσας συνάρτησης να το κάνει αυτό.
- Υπάρχουν και αρκετές άλλες ενδιαφέρουσες και χρήσιμες συναρτήσεις διαχείρισης συμβολοσειρών, όπως οι `strncpy`, `strchr`, `strstr`, κλπ. Η εντολή `man` είναι ένα μέσο για να μάθει κανείς περισσότερες πληροφορίες γι' αυτές.
- Κάποιες πιθανές υλοποιήσεις: [3]

```
char *strcpy(char *s1, const char *s2)
{  char *orig_s1 = s1;
   while (*s1++ = *s2++);
   return orig_s1; }

int strcmp(const char *s1, const char *s2)
{  for ( ; *s1 == *s2 ; s1++, s2++)
     if (! *s1)
       return 0;
   return *s1 - *s2; }
```

Πίνακες δεικτών και δείκτες σε δείκτες

- Όπως μπορούμε να έχουμε πίνακες ακεραίων, πίνακες πραγματικών αριθμών (κινητής υποδιαστολής, απλής ή διπλής ακρίβειας), ή πίνακες χαρακτήρων (συμβολοσειρές), έτσι μπορούμε να ορίσουμε στην C και πίνακες δεικτών.
- Με την ίδια λογική που το όνομα ενός πίνακα στοιχείων κάποιου τύπου μπορεί να θεωρηθεί (σχεδόν) ισοδύναμο με ένα δείκτη σε στοιχεία τέτοιου τύπου, έτσι και το όνομα ενός πίνακα δεικτών αντιστοιχεί σ' ένα δείκτη σε δείκτη σε στοιχεία του τύπου που δείχνουν τα στοιχεία του πίνακα.
- Παράδειγμα:

```
int i, j, *px[3], **ppx, s = 0;
for (i=0 ; i < 3 ; i++) {
  px[i] = malloc((i+2) * sizeof(int));
  if (px[i] == NULL)
    return -1; }
for (i=0 ; i < 3 ; i++)
  for (j=0 ; j < i+2 ; j++)
```

[3] Πώς θα υλοποιούσαμε τις `strlen` και `strcat`;

```
    *(px[i]+j) = i*j;
ppx = px;
for (i=0 ; i < 3 ; i++) {
  for (j=0 ; j < i+2 ; j++)
    s += *(*ppx)++;
  ppx++; }
```

Ποια θα είναι η τιμή της μεταβλητής s μετά την εκτέλεση των παραπάνω εντολών; [4]

Ορίσματα γραμμής εντολών

- Όταν εκτελούμε ένα πρόγραμμα, θέλουμε να έχουμε, κατά την κλήση του προγράμματος, τη δυνατότητα να δίνουμε στη γραμμή εντολών κάποια ορίσματα, τα οποία να μπορούν να περάσουν στο πρόγραμμα, για να τα επεξεργαστεί κατάλληλα.
- Στην C, η δυνατότητα ορισμάτων στη γραμμή εντολών παρέχεται μέσω των τυπικών παραμέτρων της συνάρτησης `main`.
- Ορίζοντας την `main` με δύο τυπικές παραμέτρους, μία ακέραια, συνήθως με όνομα `argc`, και μία πίνακα δεικτών σε χαρακτήρες (ή δείκτη σε δείκτη σε χαρακτήρες), συνήθως με όνομα `argv`, μέσα στη συνάρτηση, μπορούμε να έχουμε πρόσβαση στα ορίσματα με τα οποία κλήθηκε το πρόγραμμα στη γραμμή εντολής.
- Αν ένα εκτελέσιμο πρόγραμμα `myprog` κληθεί σαν

  ```
  % ./myprog first 241 third
  ```

  και η συνάρτηση `main` του προγράμματος έχει ορισθεί σαν

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

  τότε, μέσα στην `main`, η μεταβλητή `argc` έχει την τιμή 4 (όσα τα ορίσματα συν το όνομα του προγράμματος) και οι δείκτες `argv[0]`, `argv[1]`, `argv[2]` και `argv[3]` δείχνουν στην αρχή των συμβολοσειρών `"./myprog"`, `"first"`, `"241"` και `"third"`, αντίστοιχα.
- Ο δείκτης `argv[argc]` (στο προηγούμενο παράδειγμα ο `argv[4]`) είναι ο κενός δείκτης, `NULL`.
- Η δεύτερη τυπική παράμετρος της `main` μπορεί να ορισθεί είτε σαν `char *argv[]` είτε σαν `char **argv`, ισοδύναμα.
- Όταν κάποιο όρισμα στη γραμμή εντολών είναι ακέραιος αριθμός, μέσα στο πρόγραμμά μας, κατά πάσα πιθανότητα, θα μας ενδιαφέρει να έχουμε διαθέσιμη την αριθμητική τιμή του αριθμού, όχι απλώς τα ψηφία του σαν μία συμβολοσειρά.
- Η μετατροπή μίας συμβολοσειράς που τα στοιχεία της είναι δεκαδικά ψηφία στην αντίστοιχη αριθμητική τιμή γίνεται με τη συνάρτηση

[4] 15

```
int atoi(const char *s)
```

- Η `atoi` υπολογίζει την τιμή της (αριθμητικής) συμβολοσειράς `s` και την επιστρέφει στο όνομά της.
- Στο προηγούμενο παράδειγμα η κλήση `atoi(argv[2])` θα επέστρεφε την τιμή `241`.
- Όταν χρησιμοποιούμε την `atoi`, πρέπει να κάνουμε και:

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

Πολυδιάστατοι πίνακες

- Εκτός από μονοδιάστατους πίνακες, στην C, όπως και σε όλες σχεδόν τις γλώσσες προγραμματισμού, μπορούμε να ορίσουμε και πίνακες με περισσότερες της μίας διαστάσεις.
- Με τη δήλωση

```
int matrix[10][8];
```

ορίζουμε ένα δισδιάστατο πίνακα με όνομα `matrix` με `10` γραμμές και `8` στήλες.

- Η αναφορά στα στοιχεία του πίνακα γίνεται με τρόπο αντίστοιχο με αυτόν των μονοδιάστατων πινάκων. Το στοιχείο `matrix[i][j]` είναι αυτό που βρίσκεται στην `i` γραμμή και στην `j` στήλη. [5]
- Αν έχουμε ορίσει ένα δισδιάστατο πίνακα, έστω με όνομα `matrix`, τότε το `matrix[i]` (ισοδύναμα το `*(matrix+i)`) είναι ένας δείκτης στο πρώτο στοιχείο της `i` γραμμής του πίνακα. Οπότε, το `*(matrix[i]+j)` (ισοδύναμα το `*(*(matrix+i)+j)`) δεν είναι άλλο από το στοιχείο `matrix[i][j]` του πίνακα.
- Ένας δισδιάστατος πίνακας αποθηκεύεται κατά γραμμές. Αν αναθέσουμε σ' ένα δείκτη `p`, που δείχνει σε τύπο ό,τι και ο τύπος των στοιχείων του πίνακα, τη διεύθυνση του πρώτου στοιχείου του πίνακα (π.χ. `matrix[0][0]`), τότε μπορούμε να προσπελάσουμε τα περιεχόμενα του πίνακα, τη μία γραμμή μετά την άλλη, με παραστάσεις της μορφής `*(p+i)`.
- Επίσης, αφού το `*matrix` είναι το ίδιο με το `matrix[0]`, δηλαδή ένας δείκτης στο πρώτο στοιχείο της πρώτης γραμμής (αυτής με `i=0`) του πίνακα, μπορούμε να προσπελάσουμε κατά γραμμές τα περιεχόμενα του πίνακα και με παραστάσεις της μορφής `*(*matrix+i)`.
- Φυσικά, μπορούμε να ορίσουμε και πίνακες με περισσότερες των δύο διαστάσεις, αλλά, σε πρώτη φάση, δεν έχουν ιδιαίτερη προγραμματιστική χρησιμότητα.
- Αν θέλουμε να περάσουμε σε μία συνάρτηση έναν πολυδιάστατο πίνακα, τότε στην κλήση της συνάρτησης δίνουμε το όνομά του και στον ορισμό της δηλώνουμε, μαζί με το

[5] Μόνο, προσοχή και εδώ, το `i` πρέπει να κυμαίνεται, στο παράδειγμά μας, από `0` έως `9` και το `j` από `0` έως `7`.

όνομά του, και όλες τις διαστάσεις του, εκτός της πρώτης, η οποία δεν είναι υποχρεωτική. Παράδειγμα:

```
int main(void)
{  int matr[16][12];
   ....
   myfun(matr);
   ....  }

void myfun(int matr[][12])
{  ....  }
```

- Από τα προηγούμενα είναι εμφανές ότι όταν χρησιμοποιούμε πίνακες που ορίζονται στατικά, αναγκαστικά δηλώνουμε συγκεκριμένες διαστάσεις κατά τη φάση συγγραφής των προγραμμάτων μας, με αποτέλεσμα είτε να κάνουμε σπατάλη μνήμης είτε να περιοριζόμαστε στην επίλυση προβλημάτων με μικρό μέγεθος. Είναι προτιμότερο να χρησιμοποιούμε δείκτες και να κάνουμε δυναμική δέσμευση μνήμης.
- Αν σ' ένα πρόγραμμα χρειάζεται να φυλάξουμε δεδομένα σ' ένα δισδιάστατο πίνακα, αγνώστου, κατ' αρχήν, μεγέθους, αντί να δηλώσουμε κάτι σαν

  ```
  int x[10000][10000];
  ```

  μπορούμε να ορίσουμε ένα δείκτη

  ```
  int **px;
  ```

  και μετά να κάνουμε:

  ```
  px = malloc(N * sizeof(int *));
  if (px == NULL)
     return -1;
  for (i=0 ; i < N ; i++) {
     *(px+i) = malloc(M * sizeof(int));
     if (*(px+i) == NULL)
        return -1;  }
  ```

- Έτσι, όταν τα `N` και `M` γίνουν γνωστά κατά τη φάση εκτέλεσης του προγράμματος, θα δεσμεύσουμε όση ακριβώς μνήμη χρειαζόμαστε. Αφού τελειώσουμε ό,τι έχουμε να κάνουμε, μετά πρέπει να αποδεσμεύσουμε τη μνήμη ως εξής:

  ```
  for (i=0 ; i < N ; i++)
     free(*(px+i));
  free(px);
  ```

- Για να περάσουμε σε μία συνάρτηση ένα πολυδιάστατο πίνακα ορισμένο δυναμικά, αρκεί να την καλέσουμε με παράμετρο τον δείκτη μέσω του οποίου δεσμεύτηκε η μνήμη, π.χ. `fun(px);`, και να έχουμε ορίσει τη συνάρτηση με τυπική παράμετρο ένα δείκτη του

κατάλληλου τύπου. Δηλαδή:

```
void fun(int **px) { ....
```

Αρχικοποίηση πινάκων

- Όταν ορίζουμε μία μεταβλητή, μπορούμε, ως γνωστόν, να της δώσουμε και κάποια αρχική τιμή. Η δυνατότητα αυτή υπάρχει και για τους πίνακες.
- Παραδείγματα:

```
int x[7] = {5, 3, -7, 12, 0, -4, 125};
float r[] = {1.2, -4.35, 0.62e-17};
int matrix[3][5] = {
   {22, 1, -3, 8, 7},
   {-2, 0, 24, 7, -10},
   {76, 54, 12, -9, 8}
};
char arr[][2] = {{'a','b'},{'c','d'},{'e','f'}};
char mess[] = "a dog";
char buff[] = {'a',' ','d','o','g','\0'};
char *str[] = {"Hello", "world", "of", "C"};
```

- Μπορούμε, κατά τον ορισμό, να παραλείψουμε το μέγεθος ενός μονοδιάστατου πίνακα, αν αρχικοποιούμε όλα τα στοιχεία του, αφού αυτό μπορεί να προκύψει από το πλήθος των τιμών μέσα στα { και }.
- Σε πολυδιάστατους πίνακες, μπορούμε να παραλείψουμε το μέγεθος μόνο της πρώτης διάστασης.

Δείκτες σε συναρτήσεις

- Όπως μία μεταβλητή αντιστοιχεί σε μία διεύθυνση μνήμης που μπορούμε να τη φυλάξουμε σ' ένα δείκτη κατάλληλου τύπου, όπως επίσης μπορούμε να αναφερόμαστε σε πίνακες με τη διεύθυνση του πρώτου τους στοιχείου, έτσι μπορούμε να αναφερόμαστε και σε συναρτήσεις μέσω δεικτών.
- Ένας δείκτης σε συνάρτηση είναι μία μεταβλητή στην οποία μπορούμε να καταχωρήσουμε τη διεύθυνση της πρώτης από τις (συνεχόμενες) θέσεις μνήμης στις οποίες βρίσκεται ο κώδικας της συνάρτησης.
- Τους δείκτες συναρτήσεων μπορούμε να τους χειριστούμε περίπου όπως και τους άλλους δείκτες. Με τον τελεστή έμμεσης αναφοράς *, μπορούμε να αναφερθούμε σε συγκεκριμένες συναρτήσεις, μπορούμε να αναθέτουμε τις τιμές αυτών των δεικτών σε άλλους συμβατούς δείκτες, μπορούμε να τους περνάμε σαν παραμέτρους σε συναρτήσεις κλπ. Δεν έχει νόημα όμως να τους μεταβάλλουμε (π.χ. p++).

- Παράδειγμα:

  ```
  int (*funvar)(char *);
  ```

  Το funvar είναι ένας δείκτης σε συνάρτηση που επιστρέφει int και έχει μία τυπική παράμετρο τύπου char *.

- Τι διαφορά θα είχε αν γράφαμε το παρακάτω; [6]

  ```
  int *funvar(char *);
  ```

Διαχείριση ορισμάτων στη γραμμή εντολής

```
/* File: cmdlineargs.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int checkint(char *);
void reverse(char *);

int main(int argc, char *argv[])
{ int sum = 0;   /* Initialize sum of integers in command line */
  while (--argc) {         /* Loop while arguments are present */
    if(checkint(*++argv))           /* Is argument an integer? */
      sum += atoi(*argv); /* Then, sum it into the accumulator */
    else {
      reverse(argv[0]); /* Else, reverse argument and print it */
      printf("Reversed argument: %s\n", argv[0]);
    }
  }                /* Finally, print sum of integer arguments */
  printf("\nSum of integers given is: %d\n", sum);
  return 0;
}

int checkint(char *s)
{ char *start;
  while(*s == ' ' || *s == '\t') s++;    /* Eat all whitespace */
  if (*s == '-' || *s == '+') s++;             /* Eat one sign */
  start = s;                    /* Mark the begining of numbers */
  while(*s >= '0' && *s <= '9') s++;        /* Eat all numbers */
  return (*s == '\0' && start != s);
             /* Return if there where numbers and nothing else */
```

[6] Εδώ το funvar είναι το όνομα συγκεκριμένης συνάρτησης, όχι δείκτης σε συνάρτηση, που επιστρέφει δείκτη σε ακέραιο (int *) και έχει μία τυπική παράμετρο τύπου char *.

```
}

void reverse(char *s)
{ char c;
  int i, j;
  for (i=0, j=strlen(s)-1 ; i < j ; i++, j--) {
    c = s[i];  /* Visit string from start and end concurrently */
    s[i] = s[j];       /* and exchange characters in symmetric */
    s[j] = c;             /* positions until middle is reached */
  }
}
```

```
% gcc -o cmdlineargs cmdlineargs.c
% ./cmdlineargs This is a test
Reversed argument: sihT
Reversed argument: si
Reversed argument: a
Reversed argument: tset

Sum of integers given is: 0
% ./cmdlineargs And 123 another 12
Reversed argument: dnA
Reversed argument: rehtona

Sum of integers given is: 135
% ./cmdlineargs minus five -5 plus twenty two 22
Reversed argument: sunim
Reversed argument: evif
Reversed argument: sulp
Reversed argument: ytnewt
Reversed argument: owt

Sum of integers given is: 17
%
```

Πρόσθεση πολυψήφιων αριθμών

```
/* File: addnumbs.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *addnumbs(char *, char *);
```

```c
int main(int argc, char *argv[])
{ char *sum;
  if (argc != 3) {               /* Run with exactly two arguments */
    printf("Usage: %s <numb1> <numb2>\n", argv[0]); return 1; }
  if ((sum = addnumbs(argv[1], argv[2])) == NULL) { /* Compute sum */
    printf("Sorry, a memory problem occurred\n"); return 1; }
  printf("%s + %s = %s\n", argv[1], argv[2], sum);
  free(sum);                   /* Free memory malloc'ed by addnumbs */
  return 0;
}
char *addnumbs(char *s1, char *s2)
{ int i, j, k, d1, d2, sum, carry;
  char *s3, *tmp, *result;
  i = strlen(s1)-1;     /* Index to last character of first number */
  j = strlen(s2)-1;    /* Index to last character of second number */
  k = (i > j) ? (i+1) : (j+1);   /* Index to last character of sum */
  if ((s3 = malloc((k+2)*sizeof(char))) == NULL)
    return NULL;                     /* Allocate memory for result */
  s3[k+1] = '\0';         /* Proper termination of resulting string */
  carry = 0;                         /* Initial carry for addition */
  for ( ; k >= 0 ; i--, j--, k--) {
                     /* Loop till first digit of result is computed */
    d1 = (i >= 0) ? (s1[i]-'0') : 0;       /* Get i-th digit of s1 */
    d2 = (j >= 0) ? (s2[j]-'0') : 0;       /* Get j-th digit of s2 */
    sum = d1+d2+carry;                           /* Sum two digits */
    carry = sum/10;                                 /* Next carry */
    s3[k] = sum%10+'0';                /* Put k-th digit of result */
  }
  tmp = s3;                    /* Save s3 for freeing it afterwards */
  while (*s3 == '0')                /* Eat leading 0s in the result */
    s3++;
  if(*s3 == '\0')                   /* At least one digit is needed */
    s3--;
  if ((result = malloc((strlen(s3)+1)*sizeof(char))) == NULL)
    return NULL;  /* Allocate memory for result without leading 0s */
  strcpy(result, s3);                /* Copy corrected s3 to result */
  free(tmp);                                    /* Free original s3 */
  return result;
}
```

```
% gcc -o addnumbs addnumbs.c
% ./addnumbs 12345 67890
12345 + 67890 = 80235
% ./addnumbs 125 88275346771923625166
```

```
125 + 88275346771923625166 = 88275346771923625291
% ./addnumbs 999999999999999 999999
999999999999999 + 999999 = 1000000000999998
% ./addnumbs 0000000000023 00000057
0000000000023 + 00000057 = 80
% ./addnumbs 3252352362364362362366 3262363262362363622
3252352362364362362366 + 3262363262362363622 = 3255614725626724725988
% ./addnumbs 0000000 0000000000
0000000 + 0000000000 = 0
%
```