Προγραμματισμός Ι

Συναρτήσεις

Πανεπιστήμιο Πελοποννήσου
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Νικόλαος Δ. Τσελίκας

Συναρτήσεις - Εισαγωγή

- Μία **συνάρτηση** είναι ένα ανεξάρτητο τμήμα κώδικα, που εκτελεί μία ορισμένη εργασία και **προαιρετικά** επιστρέφει μία τιμή
- Η συγγραφή προγραμμάτων με χρήση συναρτήσεων που εκτελούν ανεξάρτητες εργασίες αποτελεί τη βάση του λεγόμενου **δομημένου προγραμματισμού**
- Με τη χρήση συναρτήσεων ένα πρόγραμμα χωρίζεται σε μικρότερα τμήματα, άρα ο κώδικας διαβάζεται, τροποποιείται και ελέγχεται πιο εύκολα
- Η χρήση μίας συνάρτησης δεν περιορίζεται αποκλειστικά σε ένα πρόγραμμα, π.χ. οι συναρτήσεις βιβλιοθήκης `scanf()` και `printf()` μπορούν να χρησιμοποιηθούν σε οποιοδήποτε C πρόγραμμα
- Μέχρι τώρα, η μοναδική συνάρτηση που έχουμε γράψει είναι η συνάρτηση `main()`, ενώ τώρα θα μάθετε να γράφετε δικές σας συναρτήσεις και να τις χρησιμοποιείτε στα προγράμματά σας

Δήλωση Συνάρτησης (Πρωτότυπο Συνάρτησης)

- Η **δήλωση** (ή αλλιώς **πρωτότυπο**) μίας συνάρτησης καθορίζει το **όνομα** της συνάρτησης, τον **τύπο επιστροφής** της και **μία λίστα παραμέτρων**

- Η γενική περίπτωση δήλωσης μίας συνάρτησης έχει την παρακάτω μορφή:

```
τύπος_επιστροφής όνομα_συνάρτησης(λίστα_παραμέτρων);
```

- Το `όνομα_συνάρτησης` **πρέπει να είναι μοναδικό μέσα στο πρόγραμμα**, δηλαδή να μην υπάρχει άλλη μεταβλητή ή συνάρτηση με το ίδιο όνομα

- Η δήλωση της συνάρτησης **πρέπει να τελειώνει πάντοτε με το ελληνικό ερωτηματικό ;**

Πού γράφουμε τη Δήλωση (Πρωτότυπο) μίας Συνάρτησης ???

- Η δήλωση μίας συνάρτησης μπορεί να περιέχεται σε ξεχωριστό αρχείο, το οποίο πρέπει να συμπεριληφθεί στο πρόγραμμα με την οδηγία `#include`

- Π.χ. οι δηλώσεις των συναρτήσεων `printf()` και `scanf()` βρίσκονται στο αρχείο `stdio.h` το οποίο συμπεριλαμβάνεται στο πρόγραμμα με την οδηγία `#include <stdio.h>`.

- Εναλλακτικά, αν η δήλωση μίας συνάρτησης δεν περιέχεται σε ξεχωριστό αρχείο (κάτι που κατά 99% θα συμβεί στα δικά σας προγράμματα), τότε προτείνεται η δήλωση των συναρτήσεων να γίνεται **πριν** από τη συνάρτηση `main()`

- **Το πρωτότυπο** μίας συνάρτησης **δεν είναι υποχρεωτικό**, αλλά – σε περίπτωση που παραλείπεται – **θα πρέπει να υπάρχει ο ορισμός της συνάρτησης πριν την κλήση της συνάρτησης** (το τι σημαίνει αυτό, θα το καταλάβετε καλύτερα σε λίγο...)

Επιστροφή Συνάρτησης

- Μία συνάρτηση μπορεί να επιστρέψει το πολύ μία τιμή
- Ο `τύπος_επιστροφής` μίας συνάρτησης καθορίζει τον τύπο της τιμής επιστροφής
- Ο `τύπος_επιστροφής` μπορεί να είναι οποιοσδήποτε τύπος δεδομένων της C, όπως `char`, `int`, `double`, δείκτης σε κάποιον τύπο (π.χ. δείκτη σε `float`, δείκτη σε πίνακα, δείκτη σε συνάρτηση,…) κ.α.

Μοναδικός περιορισμός στην επιστρεφόμενη τιμή μίας συνάρτησης είναι ότι **δεν μπορεί να επιστρέψει πίνακα**

- Ο τύπος επιστροφής `void` χρησιμοποιείται όταν η συνάρτηση **δεν επιστρέφει** κάποια τιμή
- Αν στη δήλωση μίας συνάρτησης ο `τύπος_επιστροφής` έχει παραληφθεί, θεωρείται ότι η συγκεκριμένη συνάρτηση επιστρέφει `int`

Παράμετροι Συνάρτησης

- Μία συνάρτηση μπορεί **προαιρετικά** να δέχεται μία λίστα παραμέτρων (`λίστα_παραμέτρων`), που χωρίζονται μεταξύ τους με κόμμα (`,`)
- Κάθε παράμετρος χαρακτηρίζεται από τον τύπο της
- Η παράμετρος μίας συνάρτησης είναι ουσιαστικά μία μεταβλητή της συνάρτησης, η οποία θα αρχικοποιηθεί, όταν θα γίνει η κλήση της
- Εάν η συνάρτηση **δεν δέχεται παραμέτρους**, τότε η λίστα παραμέτρων δηλώνεται ως `void`
- Ωστόσο, πολλοί προγραμματιστές συνηθίζουν να παραλείπουν τη λέξη `void` και να χρησιμοποιούν κενές παρενθέσεις `()`
  - Το πιθανότερο είναι αυτή η συνήθεια να προέρχεται από παλαιότερη έκδοση της γλώσσας ή την ενασχόλησή τους με άλλες γλώσσες, όπως η C++, όπου δεν χρειάζεται να προστίθεται η λέξη `void`. Όμως, όπως και στην περίπτωση της `main()`, αν θέλουμε να είμαστε σε πλήρη συμφωνία με το πρότυπο, πρέπει να χρησιμοποιούμε τη λέξη `void`
- Και αυτό γιατί, σύμφωνα με το πρότυπο, **η χρήση κενών παρενθέσεων σημαίνει ότι η συνάρτηση δέχεται έναν άγνωστο αριθμό παραμέτρων** και όχι καμία παράμετρο

Παραδείγματα δήλωσης συναρτήσεων

```
void show(char ch); /* Δήλωση μίας συνάρτησης με όνομα
show, η οποία δέχεται σαν παράμετρο έναν χαρακτήρα και δεν
επιστρέφει τίποτα. */

double show(int a, float b); /* Δήλωση μίας συνάρτησης με
όνομα show, η οποία δέχεται μία ακέραια και μία πραγματική
παράμετρο τύπου float και επιστρέφει μία πραγματική τιμή
τύπου double. */

int *show(int *ptr1, double a); /* Δήλωση μίας συνάρτησης
με όνομα show, η οποία δέχεται σαν παραμέτρους έναν δείκτη
σε ακέραιο και μία πραγματική τιμή τύπου double και
επιστρέφει έναν δείκτη σε ακέραιο.*/
```

Παρατηρήσεις

- Σημειώστε ότι τα ονόματα των παραμέτρων δεν είναι υποχρεωτικά, αρκεί ο τύπος τους, π.χ., θα μπορούσαμε να γράψουμε:

```
double show(int, int, float);
```

- Η προτίμησή μας είναι να επιλέγονται ονόματα παραμέτρων, ώστε αυτός που διαβάζει το πρωτότυπο της συνάρτησης **να παίρνει μία ιδέα για την πληροφορία που της διοχετεύεται** και **με ποια σειρά**

- Ακόμα κι αν όλες οι παράμετροι μίας συνάρτησης έχουν τον ίδιο τύπο, ο τύπος θα πρέπει να δηλώνεται σε όλες τις παραμέτρους

Π.χ., **απαγορεύεται** να γράψουμε:

```
void test(int a, b, c);
```

αντί για:

```
void test(int a, int b, int c);
```

Ορισμός Συνάρτησης

- Η γενική μορφή ορισμού μίας συνάρτησης γίνεται ως εξής:

```
τύπος_επιστροφής όνομα_συνάρτησης(λίστα_παραμέτρων)
{
      /* Σώμα Συνάρτησης */
}
```

- Η πρώτη γραμμή πρέπει να ταιριάζει με τη δήλωση της συνάρτησης, με τη διαφορά ότι δεν προστίθεται το ερωτηματικό στο τέλος της
- Ο **κώδικας** ή, αλλιώς, το **«σώμα της συνάρτησης»** περιέχει δηλώσεις μεταβλητών και εντολές ανάμεσα σε άγκιστρα
- Ο κώδικας μίας συνάρτησης εκτελείται μόνο όταν αυτή κληθεί από κάποιο σημείο του προγράμματος
- Η εκτέλεση μίας συνάρτησης τερματίζει αν κληθεί μία εντολή τερματισμού (π.χ. `return`) ή όταν εκτελεστεί η τελευταία εντολή της
- Ο ορισμός μίας συνάρτησης μπορεί να γίνει πριν ή μετά τη **`main()`**, με προτίμηση να γίνεται μετά

Χρήση μίας συνάρτησης

- Ας δούμε ένα παράδειγμα χρήσης μίας συνάρτησης:

```
#include <stdio.h>

void test(void); /* Πρωτότυπο συνάρτησης.*/

int main(void)
{
    test(); /* Κλήση συνάρτησης. */
    return 0;
}

void test(void)/* Ορισμός συνάρτησης. */
{
    /* Function body. */
    printf("In\n");
}
```

Ο μεταγλωττιστής θα πρέπει να έχει δει το πρωτότυπο της συνάρτησης **πριν γίνει κλήση της συνάρτησης**, γι' αυτό τα πρωτότυπα συναρτήσεων δηλώνονται πριν τη `main()`...

...σε αυτή την περίπτωση, οι ορισμοί των συναρτήσεων βρίσκονται συνήθως μετά τη `main()`

Αν οι ορισμοί των συναρτήσεων βρίσκονται **πριν** τη `main()` **τα πρωτότυπα των συναρτήσεων δεν είναι απαραίτητα**

Δήλωση Συνάρτησης σε ξεχωριστό αρχείο (1/2)

- Για να δηλώσετε μία συνάρτηση σε ξεχωριστό αρχείο:
  - Δημιουργήστε ένα αρχείο, π.χ., `prototype.h`
  - Αντιγράψτε το πρωτότυπο (τη δήλωση) μέσα στο αρχείο αυτό
  - Αφαιρέστε τη δήλωση απ' το αρχείο του πηγαίου κώδικα όπου την είχατε έως τώρα
  - Χρησιμοποιήστε την οδηγία `#include` για να συμπεριλάβετε το αρχείο `prototype.h` όπως παρακάτω:

Έστω ότι το αρχείο έχει όνομα `program.c`

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

int main(void)
{
    ...
}
void test(void)
{
    ...
}
```

Το αρχείο `prototype.h` θα περιέχει μόνο τη δήλωση (πρωτότυπο) της `test()` δηλ.:

```
void test(void);
```

Παρόλο που μετακινήσαμε τη δήλωση (πρωτότυπο) της συνάρτησης `test()` στο αρχείο `prototype.h`, το αρχείο `program.c` θα εξακολουθεί να περιέχει τον ορισμό της συνάρτησης `test()`

Δήλωση Συνάρτησης σε ξεχωριστό αρχείο (2/2)

- Συνήθως, όμως, εκτός απ' τις δηλώσεις (πρωτότυπα) των συναρτήσεων, μετακινούμε και τους ορισμούς αυτών σε ξεχωριστό αρχείο (και θα έχουμε συνολικά 3 αρχεία)
- Σε αυτή την περίπτωση χρειάζεται επιπλέον:
  - Να αφαιρέσετε τους ορισμούς των συναρτήσεων απ' το κύριο πρόγραμμά σας (π.χ. απ' το `program.c` στη συγκεκριμένη περίπτωση)
  - Να δημιουργήστε ένα ακόμα αρχείο, π.χ., `my_functions.c` στο οποίο θα αντιγράψετε τους ορισμούς των συναρτήσεων
  - Να χρησιμοποιήσετε την οδηγία `#include` για να συμπεριλάβετε το αρχείο `prototype.h` και μέσα στο αρχείο `my_functions.c` όπως παρακάτω:

Περιέχει τις δηλώσεις (πρωτότυπα) των συναρτήσεων, έστω `prototype.h`

```
void test(void);
```

Το κύριο πρόγραμμα, έστω `program.c`

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

int main(void)
{
    ...
}
```

```
#include "prototype.h"

void test(void)
{
    ...
}
```

Περιέχει τους ορισμούς των συναρτήσεων, έστω `my_functions.c`

Παράδειγμα (1/2)

prog1.c

```
#include <stdio.h>

int triple_me(int a);

int main(void)
{
      int result, x = 10;

      result = triple_me(x);
      printf("result = %d", result);

      return 0;
}

int triple_me(int a)
{
      return 3*a;
}
```

Μεταγλώττιση σε gcc:

```
gcc -o prog1 prog1.c
```

Παράδειγμα (2/2)

prog1.c

```c
#include <stdio.h>
#include "my_header_file.h"

int main(void)
{
	int result, x = 10;

	result = triple_me(x);
	printf("result = %d", result);

	return 0;
}
```

my_header_file.h

```c
int triple_me(int a);

/* Function Prototype */
```

my_functions.c

```c
#include "my_header_file.h"

/* Function definition */
int triple_me(int a)
{
	return 3*a;
}
```

Μεταγλώττιση σε gcc:

gcc –o prog1 prog1.c my_functions.c

Ακόμα σωστότερα, για να ελέγχουμε αν έχει ήδη συμπεριληφθεί ή όχι το header file, συνηθίζεται να χρησιμοποιούμε οδηγίες προεπεξεργαστή (π.χ. **#ifdef** ή **#ifndef**) στο **my_header_file.h**

Σύνθετες Δηλώσεις (1/2)

- Αφού είδαμε πώς ορίζουμε συναρτήσεις, ας κάνουμε μία παρένθεση για να μάθουμε **πώς αναλύουμε σύνθετες δηλώσεις**
- Αναλύουμε κάθε δήλωση **πάντα «από μέσα προς τα έξω»**
- Βρίσκουμε το όνομα της μεταβλητής και, **αν εσωκλείεται σε παρενθέσεις** `()`, **αρχίζουμε την ανάλυση από εκεί** και συνεχίζουμε με τον επόμενο τελεστή, όπου η προτεραιότητα από την υψηλότερη στη χαμηλότερη είναι:
  - α) οι **μεταθεματικοί τελεστές**, δηλ. οι παρενθέσεις `()` που υποδεικνύουν συνάρτηση και οι αγκύλες `[]` που υποδεικνύουν πίνακα
  - β) ο **προθεματικός τελεστής** `*` υποδεικνύει «δείκτη σε...»

Σύνθετες Δηλώσεις (2/2)

- Π.χ., έστω η δήλωση: `int *p[5];`

  Το όνομα p δεν περικλείεται σε παρενθέσεις κι επειδή οι [] προηγούνται του * πάμε δεξιά και έχουμε ότι το p είναι «πίνακας πέντε…», πάμε αριστερά στο * και η δήλωση γίνεται «πίνακας πέντε δεικτών σε…», προσθέτουμε και τον τύπο και καταλήγουμε στη δήλωση: «πίνακας πέντε δεικτών σε ακεραίους»

- Άλλο ένα, έστω η δήλωση: `int (*p)[5];`

  Τώρα το p περικλείεται σε παρενθέσεις, κι επειδή υπάρχει μέσα στην παρένθεση το * έχουμε ότι το p είναι «δείκτης σε…», πάμε δεξιά τώρα στις [] και η δήλωση γίνεται «δείκτης σε πίνακα πέντε…», πάμε και αριστερά και καταλήγουμε στη δήλωση: «δείκτης σε πίνακα πέντε ακεραίων»

- Άντε κι άλλο ένα, έστω η δήλωση: `int *(*p)(int);`

  Ξεκινάμε όμοια με πριν από την παρένθεση ότι το p είναι «δείκτης σε…», οι () προηγούνται του *, άρα πάμε δεξιά και έχουμε «δείκτης σε συνάρτηση με όρισμα ακέραιο…», πάμε αριστερά στον τύπο επιστροφής και καταλήγουμε στη δήλωση «δείκτης σε συνάρτηση με όρισμα ακέραιο που επιστρέφει δείκτη σε ακέραιο»

Η εντολή return (1/2)

- Η εντολή **return** χρησιμοποιείται για τον **άμεσο τερματισμό** μίας συνάρτησης, δηλ. αν η εκτέλεση του κώδικα της συνάρτησης φτάσει σε μία εντολή **return**, τότε η συνάρτηση **τερματίζεται αυτομάτως**
- Άρα, αν εκτελεστεί η εντολή **return** μέσα στη συνάρτηση **main()**, τότε **το πρόγραμμα τερματίζεται άμεσα**

```
#include <stdio.h>
int main(void)
{
        int num;
        while(1)
        {
                printf("Enter number: ");
                scanf("%d", &num);

                if(num == 2)
                        return 0; /* Τερματισμός προγράμματος. */
                else
                        printf("Num = %d\n", num);
        }
        return 0;
}
```

- Το παραπάνω πρόγραμμα τερματίζει, αν ο χρήστης εισάγει την τιμή 2, αλλιώς εμφανίζει την εισαγόμενη τιμή (παρατηρήστε ότι η τελευταία **return** δεν θα εκτελεστεί ποτέ, αφού μόνο η πρώτη **return** είναι αυτή που μπορεί να τερματίσει το πρόγραμμα, λόγω του ατέρμονου βρόχου)

Η εντολή return (2/2)

- Αν η συνάρτηση δεν έχει οριστεί να επιστρέφει κάποια τιμή (δηλ. αν ο επιστρεφόμενος τύπος της είναι `void`), τότε – για να τερματίσουμε άμεσα σε κάποιο σημείο τη συνάρτηση - γράφουμε απλά `return;`

- Επίσης, σε αυτή την περίπτωση, η `return` στο τέλος της συνάρτησης (πριν το άγκιστρο «κλεισίματος» `}`) δεν είναι απαραίτητη, αφού η συνάρτηση θα επιστρέψει αυτόματα, δεδομένου ότι τελειώνει το «σώμα» της

- Αν, όμως, η συνάρτηση έχει οριστεί να επιστρέφει κάποια τιμή, τότε η εντολή `return` **πρέπει να ακολουθείται** από κάποια τιμή

- Αυτή η τιμή επιστρέφεται στο πρόγραμμα που την κάλεσε

- Ο τύπος της τιμής που επιστρέφεται πρέπει **να είναι ίδιος** με τον τύπο που ορίστηκε να επιστρέφει η συνάρτηση στη δήλωσή της, δηλαδή στο πρωτότυπό της

Παρατηρήσεις

- Όπως είπαμε, η κύρια συνάρτηση `main()` ενός προγράμματος στη C είναι και αυτή μία συνάρτηση
- Η `main()` καλείται από το λειτουργικό σύστημα όταν αρχίζει η εκτέλεση του προγράμματος και τερματίζεται όταν τελειώνει η εκτέλεση του προγράμματος
- Η τιμή που επιστρέφει η `main()` δηλώνει τον τρόπο τερματισμού του προγράμματος και συγκεκριμένα η τιμή 0 (`return 0;`) δηλώνει τον ομαλό τερματισμό του προγράμματος, ενώ μία μη μηδενική τιμή δηλώνει συνήθως μία εσφαλμένη συνθήκη τερματισμού
- Το όνομα μίας συνάρτησης πρέπει να επιλέγεται με τέτοιο τρόπο, ώστε να περιγράφει όσο το δυνατόν καλύτερα τον σκοπό της
- Π.χ. αν θέλετε να δηλώσετε μία συνάρτηση που να υπολογίζει το άθροισμα κάποιων αριθμών, τότε ένα επιτυχημένο περιγραφικό όνομα θα μπορούσε να είναι το `sum` ή το `athroisma` και όχι ένα όνομα όπως `function`, `func`, `test`, `foufoutos` ή `lala`

Παραδείγματα Συναρτήσεων (Ι)

```
void test(int a, int b) /* Ορισμός συνάρτησης. */
{
        /* Σώμα συνάρτησης. */
        if(a == b)
                return;

        printf("%f\n", (a+b)/2.0);
        /* Δεν χρειάζεται να προστεθεί η return. */
}
```

Πού τερματίζεται η συνάρτηση `test()` ????

Η συνάρτηση `test()` συγκρίνει τις τιμές των παραμέτρων της και, αν είναι ίσες, η εκτέλεση της συνάρτησης τερματίζεται, ενώ αν είναι διαφορετικές, εμφανίζει τον μέσο όρο τους και στη συνέχεια η εκτέλεση της συνάρτησης τερματίζεται.

Παραδείγματα Συναρτήσεων (ΙΙ)

```
int test2(int a, int b) /* Ορισμός συνάρτησης. */
{
        if(a == b)
                return 0;
        else
                return 1;
}
```

Πού τερματίζεται η συνάρτηση `test2()` ????

Η συνάρτηση `test2()` συγκρίνει τις τιμές των παραμέτρων της και, αν είναι ίσες, η συνάρτηση επιστρέφει την τιμή `0` και η εκτέλεση της συνάρτησης τερματίζεται, ενώ αν οι τιμές είναι διαφορετικές, επιστρέφει την τιμή `1` και η εκτέλεση της συνάρτησης τερματίζεται.

Παραδείγματα Συναρτήσεων (ΙΙΙ)

- Όταν μία συνάρτηση επιστρέφει κάποια τιμή, θα πρέπει όλα τα δυνατά «μονοπάτια» της, να επιστρέφουν κάποια τιμή

```
float absolute(float a, float b)     /* Ορισμός συνάρτησης. */
{
        /* Σώμα συνάρτησης. */
        if(a > b)
                return a-b;
        else if (a < b)
                return b-a;
        else
                return 0;
}
```

Η ακέραια τιμή θα μετατραπεί σε δεκαδική.

Παραδείγματα Συναρτήσεων (IV)

- Ποια τιμή επιστρέφει η συνάρτηση `test3()` ????

```
int test3(void)   /* Ορισμός συνάρτησης. */
{
      return 4.9;
}
```

Η επιστρεφόμενη τιμή θα είναι **4** και όχι **4.9** (αφού η τιμή επιστροφής έχει οριστεί να είναι τύπου **int**)

Κλήση Συνάρτησης

- Όταν καλείται μία συνάρτηση, η εκτέλεση του προγράμματος συνεχίζει με την εκτέλεση του κώδικα της συνάρτησης
- Όταν τερματίζεται η συνάρτηση, η εκτέλεση του προγράμματος **επιστρέφει στο σημείο κλήσης** της συνάρτησης και συνεχίζει με την εκτέλεση της επόμενης εντολής
- Μία συνάρτηση μπορεί να κληθεί **όσες φορές είναι απαραίτητο** για τους σκοπούς του προγράμματος
- Όταν γίνεται η κλήση μίας συνάρτησης, ο μεταγλωττιστής **δεσμεύει μνήμη** για να αποθηκεύσει τις μεταβλητές που δηλώνονται στη λίστα παραμέτρων της συνάρτησης, καθώς και αυτές που δηλώνονται μέσα στο σώμα της
- Αυτή η μνήμη **δεσμεύεται** από ένα συγκεκριμένο τμήμα μνήμης που παρέχει το λειτουργικό σύστημα στο πρόγραμμα και ονομάζεται **στοίβα (stack)**
- Η **αποδέσμευση** αυτής της μνήμης **γίνεται αυτόματα** όταν τερματιστεί η εκτέλεση της συνάρτησης

Παρατηρήσεις

- Όταν χρησιμοποιείτε μία συνάρτηση βιβλιοθήκης, θα πρέπει να συμπεριλάβετε το αρχείο που περιέχει τη δήλωσή της με την οδηγία `#include`

- Για παράδειγμα, για να χρησιμοποιηθεί η συνάρτηση `strlen()` (δηλ. για να μπορέσετε να καλέσετε τη συγκεκριμένη συνάρτηση σε ένα πρόγραμμά σας), πρέπει να προστεθεί το αρχείο `string.h` με την οδηγία:

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

αλλιώς ο μεταγλωττιστής θα εμφανίσει μήνυμα λάθους για αδήλωτη συνάρτηση

Κλήση Συνάρτησης χωρίς Παραμέτρους

- Η κλήση μίας συνάρτησης που δεν δέχεται παραμέτρους σημαίνει ότι δεν της μεταβιβάζεται κάποια πληροφορία

- Η κλήση μίας συνάρτησης που δεν δέχεται παραμέτρους γίνεται γράφοντας (μέσα στο πρόγραμμα, στο σημείο που επιθυμούμε να την καλέσουμε) το όνομά της, ακολουθούμενο από κενές παρενθέσεις

Παράδειγμα κλήσης συνάρτησης χωρίς παραμέτρους που δεν επιστρέφει τιμή

```
#include <stdio.h>

void test(void); /* Πρωτότυπο συνάρτησης. */

int main(void)
{
      printf("Call_1 ");
      test(); /* Κλήση συνάρτησης. Οι παρενθέσεις είναι κενές,
γιατί η συνάρτηση δεν δέχεται παραμέτρους. */
      printf("Call_2 ");
      test(); /* Δεύτερη κλήση. */
      return 0;
}

void test(void) /* Ορισμός συνάρτησης. */
{
      /* Σώμα συνάρτησης. */
      int i;
      for(i = 0; i < 2; i++)
            printf("In ");
}
```

Έξοδος: `Call_1 In In Call_2 In In`

Παράδειγμα κλήσης συνάρτησης χωρίς παραμέτρους που επιστρέφει τιμή

```
#include <stdio.h>

int test(void); /* Πρωτότυπο συνάρτησης. */

int main(void)
{
      int sum;

      sum = test(); /* Κλήση συνάρτησης. Η τιμή που επιστρέφει η
test() αποθηκεύεται στη sum. */
      printf("%d\n", sum);
      return 0;
}

int test(void) /* Ορισμός συνάρτησης. */
{
      int i = 10, j = 20;
      return i+j;
}
```

Έξοδος: 30

Κλήση Συνάρτησης με Παραμέτρους (Ι)

- Η κλήση μίας συνάρτησης που δέχεται παραμέτρους σημαίνει ότι στη συνάρτηση μεταβιβάζεται πληροφορία μέσω των ορισμάτων της

Η διαφορά μεταξύ παραμέτρου και ορίσματος είναι ότι ο όρος παράμετρος αναφέρεται στις μεταβλητές που εμφανίζονται στη δήλωση και στον ορισμό της συνάρτησης, ενώ ο όρος όρισμα αναφέρεται στις εκφράσεις που περιέχονται στην κλήση της συνάρτησης

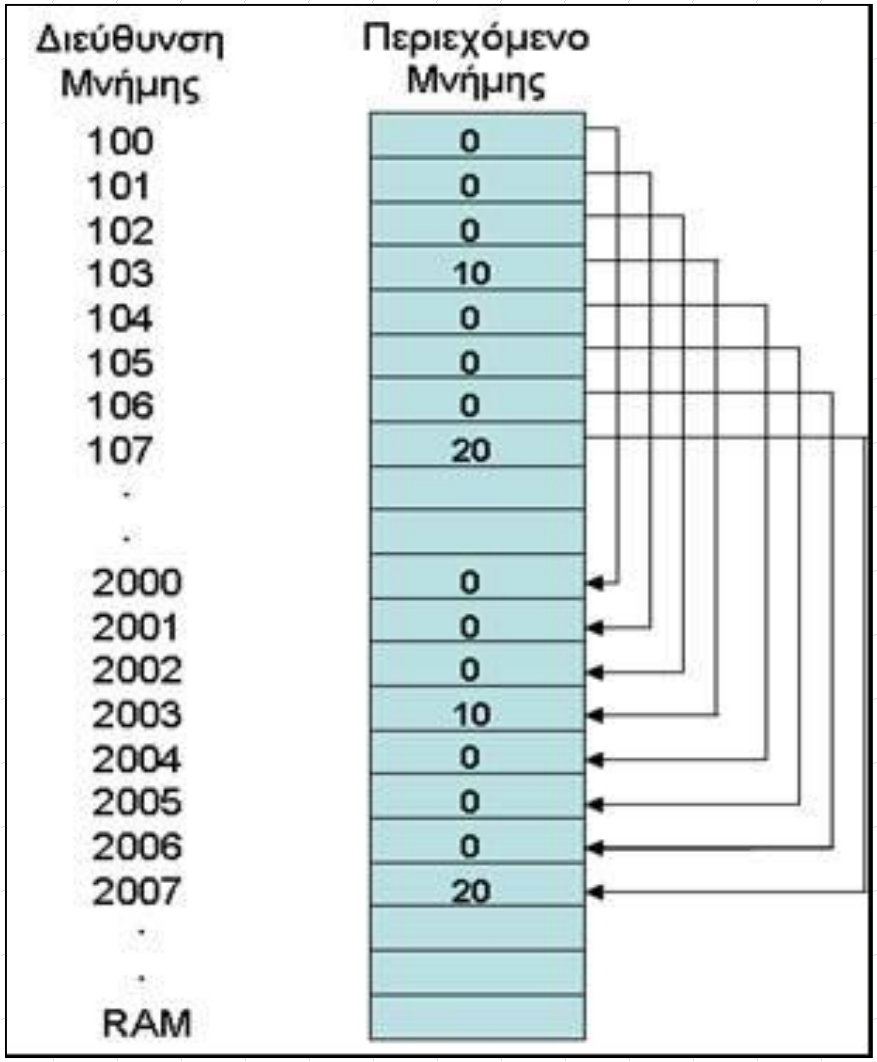
```
#include <stdio.h>

int test(int x, int y);
int main(void)
{
      int sum, a = 10, b = 20;

      sum = test(a, b); /* Οι μεταβλητές a και b είναι τα
ορίσματα της συνάρτησης. */
      printf("%d\n", sum);
      return 0;
}

int test(int x, int y) /* Οι μεταβλητές x και y είναι οι
παράμετροι της συνάρτησης. */
{
      return x+y;
}
```

Κλήση Συνάρτησης με Παραμέτρους (ΙΙ)

- Η κλήση της συνάρτησης γίνεται γράφοντας το όνομά της και μέσα σε παρενθέσεις τη λίστα ορισμάτων
- Ο τύπος του κάθε ορίσματος μπορεί να είναι μία οποιαδήποτε έγκυρη έκφραση της C, όπως π.χ. μία σταθερά, μία μεταβλητή, μία μαθηματική ή λογική έκφραση ή ακόμη και μία άλλη συνάρτηση με τιμή επιστροφής
- Το πλήθος των ορισμάτων και οι τύποι τους πρέπει να ταιριάζουν με τη δήλωση της συνάρτησης
- Π.χ. αν μία συνάρτηση έχει δηλωθεί να έχει δύο ακέραιες παραμέτρους, τότε στην κλήση της συνάρτησης πρέπει να της διοχετεύονται υποχρεωτικά δύο ακέραιες τιμές, και όχι κάτι διαφορετικό
- Ο μεταγλωττιστής ελέγχει αν το πλήθος και ο τύπος των παραμέτρων συμφωνούν με τη δήλωση της συνάρτησης
- Αν δεν υπάρχει συμφωνία είτε στο πλήθος είτε στον τύπο των παραμέτρων, τότε ενημερώνει τον προγραμματιστή με ένα λάθος μεταγλώττισης ή μήνυμα προειδοποίησης
- Όταν καλείται μία συνάρτηση οι τιμές της λίστας των παραμέτρων εκχωρούνται μία-προς-μία στις παραμέτρους της συνάρτησης

Παράδειγμα

```c
#include <stdio.h>

int test(int x, int y);

int main(void)
{
        int sum, a = 10, b = 20;

        sum = test(a, b);
        printf("%d\n", sum);
        return 0;
}

int test(int x, int y)
{
        return x+y;
}
```



Επεξήγηση Παραδείγματος

- Όταν εκτελείται το προηγούμενο πρόγραμμα, ο μεταγλωττιστής δεσμεύει 8 θέσεις μνήμης (π.χ. 100-107) για την αποθήκευση των τιμών (10 και 20) των ακέραιων μεταβλητών **a** και **b** αντίστοιχα
- Όταν καλείται η συνάρτηση **test()** ο μεταγλωττιστής δεσμεύει άλλες 8 θέσεις μνήμης (π.χ. 2000-2007) για την αποθήκευση των ακέραιων **x** και **y**, που είναι οι παράμετροι της συνάρητησης
- Στη συνέχεια, αντιγράφει τις τιμές των ορισμάτων **a** και **b** (δηλ. 10 και 20) στις αντίστοιχες θέσεις μνήμης των παραμέτρων **x** και **y**
- Όπως φαίνεται λοιπόν οι διευθύνσεις μνήμης των **παραμέτρων** (μεταβλητές **x** και **y**) **είναι διαφορετικές** από τις διευθύνσεις μνήμης των **ορισμάτων** (μεταβλητές **a** και **b**)
- Επομένως, οποιαδήποτε αλλαγή γίνει στις τιμές των **x** και **y** δεν επηρεάζει τις τιμές των **a** και **b**, άλλο αν, τα περιεχόμενα των διευθύνσεων μνήμης αμέσως μετά την αντιγραφή είναι τα ίδια
- Όταν τερματιστεί η εκτέλεση της συνάρτησης **test()** γίνεται αυτόματη αποδέσμευση της μνήμης (π.χ. 2000-2007) που είχε δεσμευτεί για τις παραμέτρους **x** και **y**

Μεταβίβαση Τιμών σε μία Συνάρτηση (Ι)

- Ένας είναι ο δυνατός τρόπος μεταβίβασης τιμών σε μία συνάρτηση στη C, η **κλήση μέσω τιμής** *(call by value)*
- Όταν γίνεται κλήση συνάρτησης **μέσω τιμής**, τότε στη συνάρτηση διοχετεύονται **οι τιμές** των ορισμάτων στη συνάρτηση και αντιγράφονται στις παραμέτρους της συνάρτησης
- Οποιαδήποτε αλλαγή γίνει στις τιμές των παραμέτρων της συνάρτησης **δεν επηρεάζει** τις τιμές των ορισμάτων που **διοχετεύθηκαν** στη συνάρτηση, γιατί οι αλλαγές γίνονται σε **διαφορετικές** διευθύνσεις μνήμης (όπως είδαμε και στο προηγούμενο παράδειγμα) αφού η συνάρτηση δέχεται τις τιμές των ορισμάτων της σε **αντίγραφα των πρωτότυπων μεταβλητών**
- **Μοναδική εξαίρεση** σε αυτόν τον κανόνα αποτελούν **οι πίνακες**
  - Όταν ένας πίνακας μεταβιβάζεται σε μία συνάρτηση, δεν δημιουργείται αντίγραφο του πίνακα, αλλά μεταβιβάζεται η διεύθυνση του πρώτου του στοιχείου

Μεταβίβαση Τιμών σε μία Συνάρτηση (ΙΙ)

- Αν επιθυμούμε μία συνάρτηση να μπορεί να αλλάξει την τιμή κάποιου ορίσματος, πρέπει να μεταβιβάσουμε στη συνάρτηση τη διεύθυνση μνήμης του ορίσματος και όχι την τιμή του

- Επομένως, αφού η συνάρτηση θα έχει πρόσβαση στη διεύθυνση μνήμης του ορίσματος, θα μπορεί να μεταβάλλει την τιμή του

- Σημειώνεται ότι σε αρκετά κείμενα/βιβλία κτλ. η μεταβίβαση της διεύθυνσης ενός ορίσματος αναφέρεται ως διαφορετικός τρόπος κλήσης με την ονομασία *κλήση μέσω αναφοράς (call by reference)*

- Ωστόσο, είναι ίδιος με την κλήση μέσω τιμής, αφού πάλι η συνάρτηση δεν μπορεί να αλλάξει την τιμή της πρωτότυπης μεταβλητής

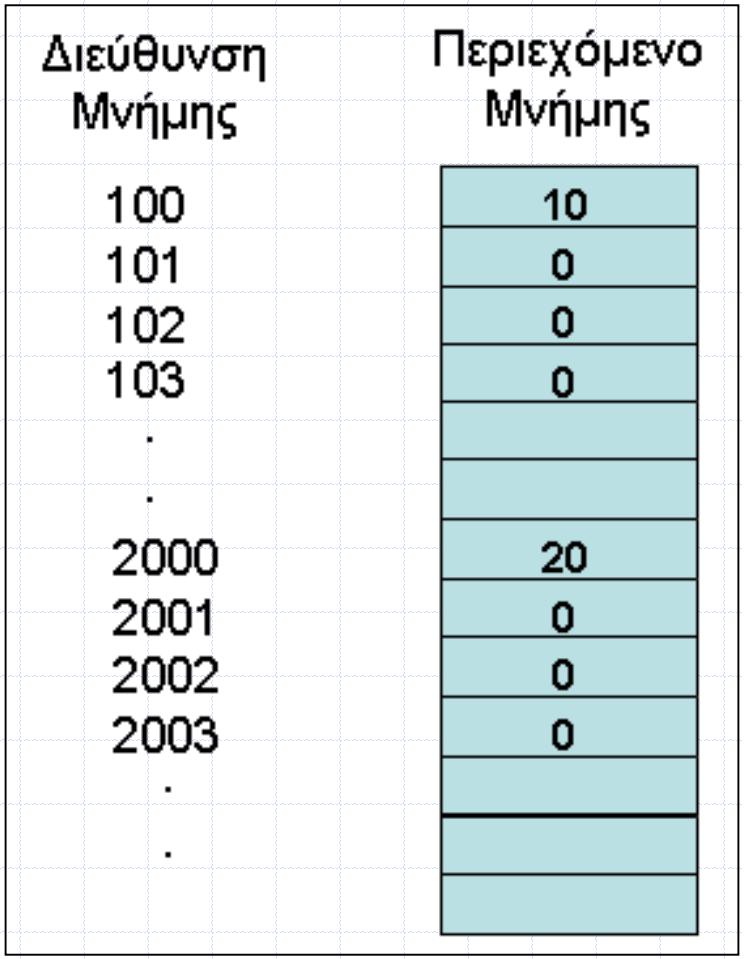
Παρατηρήσεις

- Αφού μία συνάρτηση δεν μπορεί να επιστρέψει περισσότερες από μία τιμές (θυμηθείτε ότι μία συνάρτηση επιστρέφει από καμία έως – το πολύ – μία τιμή), η μεταβίβαση της διεύθυνσης μνήμης του ορίσματος αποτελεί τον πιο ευέλικτο τρόπο για την τροποποίηση των τιμών πολλών μεταβλητών

- Σημειώστε επίσης ότι, όταν γίνεται κλήση μίας συνάρτησης, επιτρέπεται να γίνει συνδυασμός των δύο μεθόδων, δηλαδή για κάποια ορίσματα να μεταβιβαστούν οι τιμές τους ενώ για κάποια άλλα να μεταβιβαστούν οι αντίστοιχες διευθύνσεις μνήμης τους

Παράδειγμα κλήσης συνάρτησης με μεταβίβαση της τιμής του ορίσματος

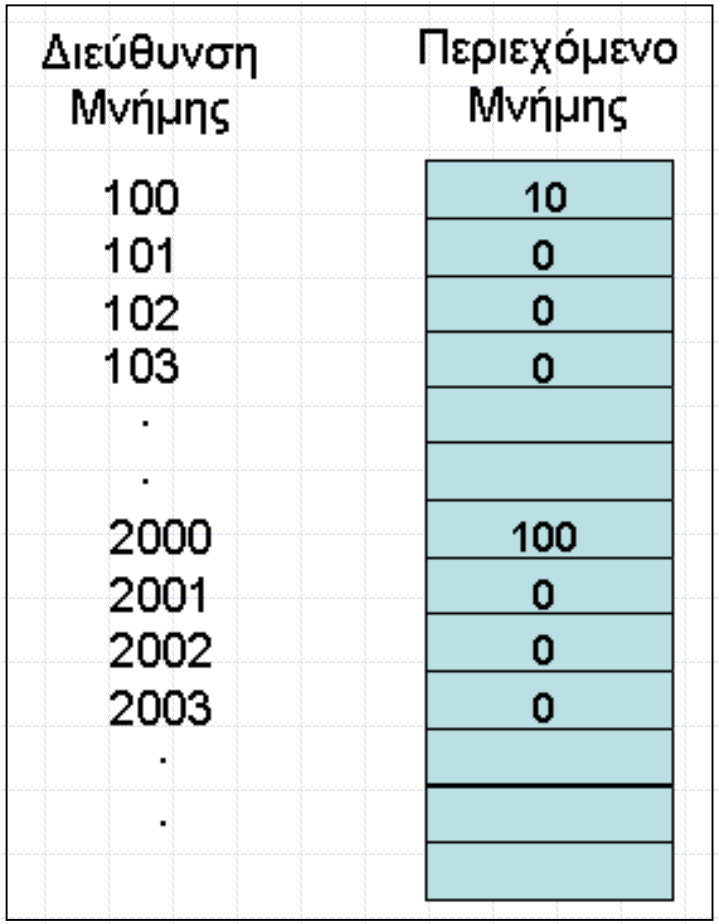
```c
#include <stdio.h>

void function(int a);

int main(void)
{
        int i = 10;

        function(i);

        printf("Val = %d\n",i);

        return 0;
}

void function(int a)
{
        a = 20;
}
```



Έξοδος: `Val = 10`

Παράδειγμα κλήσης συνάρτησης με μεταβίβαση της διεύθυνσης μνήμης του ορίσματος

```c
#include <stdio.h>

void function(int* ptr1);

int main(void)
{
        int i;
        int* ptr;

        i = 10;
        ptr = &i;

        function(ptr);
/* θα μπορούσαμε να μην δηλώσουμε
τον δείκτη ptr και να γράψουμε
κατευθείαν function(&i); */

        printf("Val = %d\n",i);

        return 0;
}

void function(int* ptr1)
{
        *ptr1 = 20;
}
```



Έξοδος: `Val = 20`

Παράδειγμα κλήσης συνάρτησης με μεταβίβαση και τιμής αλλά και διεύθυνσης μνήμης για διαφορετικά ορίσματα

```c
#include <stdio.h>

void function(int* ptr1,int a);

int main(void)
{
        int i = 100,j = 200;
        int* ptr;

        ptr = &i;
        function(ptr,j);
/* Θα μπορούσαμε να μην δηλώσουμε
 τον δείκτη ptr και να γράψουμε
κατευθείαν function(&i,j); */

        printf("%d %d\n",i,j);
        return 0;
}

void function(int* ptr1,int a)
{
        *ptr1 = 300;
        a = 400;
}
```

Έξοδος: 300 200

Διεύθυνση Μνήμης	Περιεχόμενο Μνήμης
152	100
153	0
154	0
155	0
156	200
157	0
158	0
159	0
.	
.	
.	
2000	152
2001	0
2002	0
2003	0
2004	200
2005	0
2006	0
2007	0
.	
.	

Παρατηρήσεις

- Είπαμε ήδη ότι σε αρκετά κείμενα/βιβλία κτλ. η μεταβίβαση της διεύθυνσης ενός ορίσματος αναφέρεται (κακώς) ως διαφορετικός τρόπος κλήσης με την ονομασία *κλήση μέσω αναφοράς (call by reference)*
- Με τα προηγούμενα παραδείγματα καταλάβατε ότι **τελικά πρόκειται για κλήση μέσω τιμής**, αφού και σ' αυτή την περίπτωση η συνάρτηση δεν μπορεί να αλλάξει την τιμή της πρωτότυπης μεταβλητής???
- Είστε σίγουροι???
- Ή μήπως πιστεύετε ότι στο διπλανό παράδειγμα η τιμή του `ptr` έχει αλλάξει μετά την κλήση της `test()`???

Φυσικά και όχι!!

Παρόλο που η τιμή του `p` άλλαξε και ο `p` δείχνει στο `j`, ο `ptr` παραμένει ως είχε, αφού μόνο η τιμή του `ptr` απλώς αντιγράφτηκε στον `p`

```c
#include <stdio.h>

void test(int *p);

int main(void)
{
        int *ptr, i;

        ptr = &i;
        printf("%p\n", ptr);

        test(ptr);
        printf("%p\n", ptr);
        return 0;
}

void test(int *p)
{
        int j;
        p = &j;
}
```

Παραδείγματα (Ι)

- Κατά την κλήση των συναρτήσεων **scanf()** και **printf()** μεταβιβάζεται η τιμή ή η διεύθυνση μνήμης του ορίσματος στο παρακάτω παράδειγμα???

```
#include <stdio.h>
int main(void)
{
        int a;

        scanf("%d", &a);
        printf("Value: %d\n", a);
        return 0;
}
```

Απάντηση:

scanf(): μεταβιβάζεται η διεύθυνση μίας μεταβλητής

printf(): μεταβιβάζεται η τιμή μίας μεταβλητής

Παραδείγματα (ΙΙ)

- Δημιουργήστε δύο συναρτήσεις που να υπολογίζουν το τετράγωνο και τον κύβο ενός ακεραίου, αντίστοιχα. Στη συνέχεια γράψτε ένα πρόγραμμα το οποίο να διαβάζει έναν ακέραιο και να εμφανίζει το άθροισμα του τετραγώνου και τον κύβο του ακεραίου, με χρήση των συναρτήσεων.

```c
#include <stdio.h>

int square(int a);
int cube(int a);

int main(void)
{
	int i, j, k;

	printf("Enter number: ");
	scanf("%d", &i);

	j = square(i);
	k = cube(i);
	printf("sum = %d\n", j+k); /* Without declaring the j and k variables, we could write printf("sum = %d\n", square(i)+cube(i)); */
	return 0;
}

int square(int a)
{
	return a*a;
}

int cube(int a)
{
	return a*a*a;
}
```

Παραδείγματα (ΙΙΙ)

- Δημιουργήστε μία συνάρτηση που να δέχεται σαν παράμετρο έναν ακέραιο αριθμό και έναν χαρακτήρα και να εμφανίζει τον χαρακτήρα τόσες φορές όσες και η τιμή του ακεραίου

  Στη συνέχεια γράψτε ένα πρόγραμμα το οποίο να διαβάζει έναν ακέραιο αριθμό και έναν χαρακτήρα και να εμφανίζει τον χαρακτήρα τόσες φορές όσο και ο ακέραιος (με χρήση της συνάρτησης)

```c
#include <stdio.h>

void show_char(int num, char ch);

int main(void)
{
	char ch;
	int i;

	printf("Enter character: ");
	scanf("%c", &ch);

	printf("Enter number: ");
	scanf("%d", &i);

	show_char(i, ch);
	return 0;
}

void show_char(int num, char ch)
{
	int i;

	for(i = 0; i < num; i++)
		printf("%c", ch);
}
```

Παραδείγματα (IV)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>

int f(int a);

int main(void)
{
        int i = 10;

        printf("%d\n", f(f(f(i))));
        return 0;
}

int f(int a)
{
        return a+1;
}
```

Έξοδος: 13

Παραδείγματα (V)

- Δημιουργήστε μία συνάρτηση που να δέχεται σαν παράμετρο έναν ακέραιο αριθμό (n) και να επιστρέφει την τιμή της παράστασης:
  $1^3 + 2^3 + 3^3 + \ldots + n^3$

  Στη συνέχεια γράψτε ένα πρόγραμμα το οποίο να διαβάζει έναν θετικό ακέραιο αριθμό μέχρι 1000 και να εμφανίζει την τιμή της παραπάνω παράστασης με χρήση της συνάρτησης

```c
#include <stdio.h>

double sum_cube(int num);

int main(void)
{
	int i;

	do
	{
		printf("Enter number: ");
		scanf("%d", &i);
	} while(i < 0 || i > 1000);
	printf("Result = %.0f\n", sum_cube(i));
	return 0;
}

double sum_cube(int num)
{
	int i;
	double sum; /* It is declared as
		       double in order to store
		       larger numbers. */
	sum = 0;
	for(i = 1; i <= num; i++)
		sum += i*i*i;

	return sum;
}
```

Εμβέλεια Μεταβλητών

- **Εμβέλεια** μίας μεταβλητής καλείται το τμήμα του προγράμματος, στο οποίο η μεταβλητή είναι **προσβάσιμη**, ή αλλιώς λέμε ότι **είναι «ορατή»**

- Η εμβέλεια μίας μεταβλητής εξαρτάται από το σημείο δήλωσής της μέσα στο πρόγραμμα

- Τα διαφορετικά είδη μεταβλητών βάσει της εμβέλειάς των είναι:
  - Οι **τοπικές** μεταβλητές (**local** variables)
  - Οι **καθολικές** μεταβλητές (**global** variables)

Τοπικές Μεταβλητές (local)

- Μία μεταβλητή που δηλώνεται στο σώμα μίας συνάρτησης ονομάζεται τοπική μεταβλητή (local)
- Η εμβέλεια μίας τοπικής μεταβλητής περιορίζεται στη συνάρτηση όπου δηλώνεται, γεγονός που σημαίνει ότι οι υπόλοιπες συναρτήσεις του προγράμματος δεν έχουν πρόσβαση σε αυτή τη μεταβλητή, άρα δεν μπορούν να τροποποιήσουν την τιμή της
- Αφού μία τοπική μεταβλητή δεν είναι ορατή έξω από τη συνάρτηση στην οποία δηλώνεται, μπορούμε να δηλώσουμε μεταβλητές με το ίδιο όνομα σε άλλες συναρτήσεις
- Σε πολλά από τα επόμενα προγράμματα θα δηλώνονται – επίτηδες – τοπικές μεταβλητές σε συναρτήσεις που θα έχουν το ίδιο όνομα με τοπικές μεταβλητές δηλωμένες στη συνάρτηση `main()` κυρίως για να σας γίνει ακόμα πιο ξεκάθαρο ότι αυτές οι μεταβλητές δεν έχουν καμία σχέση μεταξύ τους, παρόλο που έχουν το ίδιο όνομα

Παρατηρήσεις

- Υπενθυμίζεται ότι **μία τοπική μεταβλητή πρέπει να αρχικοποιηθεί πριν χρησιμοποιηθεί** μέσα στη συνάρτηση, γιατί ο μεταγλωττιστής της αναθέτει μία τυχαία αρχική τιμή (αυτή η τιμή συνήθως ονομάζεται «**σκουπίδια**») και δεν αναθέτει την τιμή 0

- Οι παράμετροι που δηλώνονται στην δήλωση μίας συνάρτησης θεωρούνται και αυτές τοπικές μεταβλητές της συνάρτησης (εννοείται ότι αυτές δεν χρειάζεται να αρχικοποιηθούν, αφού αρχικοποιούνται κατά την κλήση της συνάρτησης)

Παραδείγματα (I)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>

void test(void);

int main(void)
{
        int i = 10;

        test();
        printf("I_main = %d\n", i);
        return 0;
}

void test(void)
{
        int i = 200;
        printf("I_test = %d\n", i);
}
```

Output:

```
I_test = 200
I_main = 10
```

Και τι θα συμβεί αν αφαιρέσουμε τη δήλωση `int i = 200;` στη συνάρτηση `test()`?

Αφού οι δύο μεταβλητές `i` δεν σχετίζονται μεταξύ τους, ο μεταγλωττιστής θα εμφανίσει μήνυμα λάθους ότι η μεταβλητή `i` στη συνάρτηση `test()` δεν έχει δηλωθεί

Παραδείγματα (ΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>

void test(int i, int j);

int main(void)
{
      int i = 100, j = 100;

      test(i, j);
      printf("%d %d\n", i, j);
      return 0;
}

void test(int i, int j)
{
      int a = 2000; /* The local variables of the function are a, i
and j. */
      i = j = a;
}
```

Έξοδος: `Values: 100 100`

Στατικές Μεταβλητές (static)

- Κάθε φορά που καλείται μία συνάρτηση ο μεταγλωττιστής **δεσμεύει μνήμη** για τη δημιουργία των **τοπικών μεταβλητών** της συνάρτησης
- Η **μνήμη αυτή αποδεσμεύεται**, όταν τελειώνει η εκτέλεση της συνάρτησης
- Δηλαδή, κάθε τοπική μεταβλητή:
  - δημιουργείται (όταν καλείται η συνάρτηση)
  - καταστρέφεται (όταν τερματίζεται η συνάρτηση)
  - και δημιουργείται εκ νέου (όταν κληθεί ξανά η συνάρτηση)
- Αυτό σημαίνει ότι **μία τοπική μεταβλητή δεν διατηρεί την τιμή της ανάμεσα στις κλήσεις της συνάρτησης**
- Αν θέλουμε μία τοπική μεταβλητή να διατηρεί την τιμή της ανάμεσα στις κλήσεις της συνάρτησης, πρέπει να ορίζεται **σαν στατική** με τη λέξη `static`
- Μία στατική (`static`) μεταβλητή αποκτάει αρχική τιμή μόνο την πρώτη φορά που καλείται η συνάρτηση ενώ στις επόμενες κλήσεις της συνάρτησης η μεταβλητή διατηρεί την τελευταία τιμή της και δεν αρχικοποιείται πάλι

Παραδείγματα (Ι)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>

void test(void);

int main(void)
{
        test();
        test();
        test();
        return 0;
}

void test(void)
{
        static int i = 100;
        int j = 0;

        i++;
        j++;

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

Έξοδος:
```
101 1
102 1
103 1
```

Παραδείγματα (ΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>

void test(void);

int main(void)
{
        test();
        test();
        test();
        return 0;
}

void test(void)
{
        static int i;
        int j = 0;

        i = 100;

        i++;
        j++;

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

Έξοδος: 101 1
101 1
101 1

(δηλ. δεν έχει νόημα που δηλώσαμε την i ως static)

ΠΑΡΑΤΗΡΗΣΕΙΣ

Μία στατική μεταβλητή πρέπει να αρχικοποιείται όταν δηλώνεται, έτσι ώστε στις επόμενες κλήσεις της συνάρτησης να διατηρεί την τελευταία τιμή της και να μην χρειάζεται να αρχικοποιηθεί ξανά

Παρατηρήσεις (Ι)

- Αφού η μνήμη μίας απλής τοπικής μεταβλητής αποδεσμεύεται, μία συνάρτηση δεν πρέπει να επιστρέφει τη διεύθυνσή της, π.χ.

```
#include <stdio.h>

int *test(void);

int main(void)
{
        int *ptr;
        int j;

        ptr = test();
        printf("%d\n", *ptr);

        j = *ptr;
        printf("%d\n", j);
        return 0;
}

int *test(void)
{
        int i = 10;
        return &i;
}
```

Όταν καλείται η **test()**, ο μεταγλωττιστής δεσμεύει μνήμη για την τοπική μεταβλητή **i**. Η εντολή **return &i;** επιστρέφει τη διεύθυνση μνήμης της.

Όμως, αυτή η μνήμη αποδεσμεύεται όταν τερματίζει η εκτέλεση της συνάρτησης.

Επομένως, μπορεί να αποθηκευτούν νέα δεδομένα σε αυτή τη μνήμη και να χαθεί η τιμή 10.

Άρα, το πρόγραμμα αντί για τις τιμές 10 και 10 μπορεί να εμφανίσει 10 και μία άλλη τυχαία τιμή ή ακόμα και δύο τυχαίες τιμές.

Παρατηρήσεις (ΙΙ)

- Αν, στο προηγούμενο παράδειγμα, η τοπική μεταβλητή `i` είχε δηλωθεί ως `static`, η συνάρτηση θα μπορούσε να επιστρέφει τη διεύθυνση της `i`, δεδομένου ότι η μνήμη που έχει δεσμευτεί για μία `static` μεταβλητή δεν αποδεσμεύεται όταν τερματιστεί η συνάρτηση

ΠΡΟΣΟΧΗ ΛΟΙΠΟΝ!!!

Μην επιστρέφετε τη διεύθυνση μίας τοπικής μεταβλητής, εκτός αν έχει δηλωθεί ως `static`

Καθολικές Μεταβλητές (global)

- Μία μεταβλητή που δηλώνεται έξω από οποιαδήποτε συνάρτηση ονομάζεται καθολική (global) μεταβλητή
- Η εμβέλεια μίας καθολικής μεταβλητής εκτείνεται από το σημείο της δήλωσής της μέχρι το τέλος του αρχείου στο οποίο δηλώνεται
- Επομένως, όλες οι συναρτήσεις που ορίζονται μετά από το σημείο δήλωσης της καθολικής μεταβλητής έχουν πρόσβαση σε αυτήν και μπορούν να τροποποιήσουν την τιμή της

Παρατηρήσεις

- Συνήθως, μία μεταβλητή δηλώνεται σαν καθολική όταν χρησιμοποιείται **σε πολλά τμήματα** του προγράμματος
- Συστήνεται να επιλέγετε περιγραφικά ονόματα για τις καθολικές μεταβλητές, ώστε το πρόγραμμα να διαβάζεται πιο εύκολα, π.χ., μην χρησιμοποιείτε ονόματα τα οποία συνήθως δίνονται σε τοπικές μεταβλητές ή δεν έχουν κάποιο ιδιαίτερο νόημα, π.χ. `i`, `k`, `foufoutos` ή `lala`, αλλά να επιλέγετε ονόματα που **περιγράφουν όσο το δυνατόν καλύτερα τον σκοπό της**
- Να δίνετε αρχική τιμή (να κάνετε δηλ. ρητή αρχικοποίηση) σε μία καθολική μεταβλητή, αμέσως όταν τη δηλώνετε
- Αν σε μία καθολική μεταβλητή δεν έχει ανατεθεί αρχική τιμή, ο μεταγλωττιστής την αρχικοποιεί με `0`
- Γενικότερα, **προτιμούμε να αποφεύγουμε τις δηλώσεις καθολικών μεταβλητών**, διότι – αν συμβεί κάποιο λάθος – αρκετές φορές είναι δύσκολο να εντοπιστεί το σημείο που έγινε το λάθος (π.χ. ποια απ' όλες τις συναρτήσεις ανάθεσε λανθασμένη τιμή στην καθολική μεταβλητή)

Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>

void add(void);
void sub(void);

int glob = 10;

int main(void)
{
        add();
        printf("Val = %d\n", glob);
        sub();
        printf("Val = %d\n", glob);
        return 0;
}

void add(void)
{
        glob++;
}

void sub(void)
{
        glob--;
}
```

Έξοδος: Val = 11
Val = 10

Παρατηρήσεις

- Μία τοπική μεταβλητή είναι διαφορετική από μία καθολική μεταβλητή, ακόμα και αν έχουν το ίδιο όνομα

- Π.χ. στο παρακάτω πρόγραμμα η τοπική μεταβλητή a που δηλώνεται στην test() είναι διαφορετική από την καθολική μεταβλητή a

```
#include <stdio.h>

void test(void);

int a = 100;

int main(void)
{
        test();
        printf("Val = %d\n", a);
        return 0;
}

void test(void)
{
        int a;
        a = 2000;
}
```

Έξοδος: Val = 100

Εξωτερικές Καθολικές Μεταβλητές (extern global)

- Ο κώδικας ενός προγράμματος μπορεί να περιέχεται σε περισσότερα από ένα αρχεία, γεγονός πιθανό, ιδίως σε περίπτωση μεγάλων προγραμμάτων

- Σε ένα τέτοιο πρόγραμμα υπάρχει το ενδεχόμενο μία μεταβλητή που χρησιμοποιείται σε ένα αρχείο να πρέπει να χρησιμοποιηθεί και σε κάποιο άλλο αρχείο

- Σε μία τέτοια περίπτωση η μεταβλητή που πρέπει να είναι «ορατή» σε περισσότερα από ένα αρχεία δηλώνεται σαν καθολική στο αρχείο που χρησιμοποιείται περισσότερο (συνήθως) ενώ σε όποιο άλλο αρχείο χρησιμοποιείται αυτή η μεταβλητή πρέπει να δηλώνεται σαν καθολική με χρήση της λέξης `extern`

  Π.χ. η εντολή:

```
extern int size;
```

  ενημερώνει τον μεταγλωττιστή ότι η ακέραια μεταβλητή με όνομα `size` δεν δηλώνεται σε αυτό το αρχείο, αλλά σε κάποιο άλλο αρχείο του προγράμματος

- Σημειώνεται, ότι αυτή η μεταβλητή μπορεί να χρησιμοποιηθεί και, αν χρειαστεί, να τροποποιηθεί η τιμή της σε οποιοδήποτε από τα αρχεία στα οποία περιέχεται και όχι μόνο στο αρχείο δήλωσής της

Δήλωση (πρωτότυπο) Συνάρτησης με παράμετρο Πίνακα

- Όταν θέλουμε να δηλώσουμε μία συνάρτηση που έχει **σαν παράμετρο έναν πίνακα**, τότε (στη δήλωση της συνάρτησης) στις παραμέτρους της γράφουμε το όνομα του πίνακα ακολουθούμενο από κενές αγκύλες

Π.χ. με την ακόλουθη δήλωση, η συνάρτηση με όνομα `test` δέχεται σαν παράμετρο έναν πίνακα ακεραίων και δεν επιστρέφει κάποια τιμή

```
void test(int arr[]); /* Δήλωση συνάρτησης που
έχει σαν παράμετρο έναν πίνακα ακεραίων και δεν
επιστρέφει τίποτα. */
```

- Όταν καλούμε μία συνάρτηση που έχει παράμετρο πίνακα, κατά την κλήση της συνάρτησης γράφουμε **ως όρισμα μόνο το όνομα του πίνακα, χωρίς τις αγκύλες**, π.χ.:

```
test(arr);
```

Κλήση Συνάρτησης με παράμετρο Πίνακα (Ι)

- Όταν το όνομα του πίνακα χρησιμοποιείται ως όρισμα μίας συνάρτησης, χρησιμοποιείται πάντοτε ως δείκτης. Ουσιαστικά, ως όρισμα περνάμε τη διεύθυνση του πρώτου στοιχείου του πίνακα

- Ξαναλέμε λοιπόν, κατά την **κλήση** μίας συνάρτησης με όρισμα πίνακα, γράφουμε **μόνο** το **όνομα** του πίνακα, **χωρίς τις αγκύλες**, όπως φαίνεται στο παρακάτω παράδειγμα

```
void function(int arr[]); /* Δήλωση συνάρτησης που έχει σαν
παράμετρο έναν πίνακα ακεραίων και δεν επιστρέφει τίποτα. */

int main(void)
{
	int pin[100];

	function(pin); /* Στην κλήση της συνάρτησης γράφουμε το
όνομα του πίνακα χωρίς τις αγκύλες. */
	return 0;
}

void function(int arr[])
{
	/* Σώμα συνάρτησης. */
}
```

Κλήση Συνάρτησης με παράμετρο Πίνακα (ΙΙ)

- Όπως είδαμε στο κεφάλαιο των δεικτών, το όνομα ενός πίνακα είναι δείκτης στο πρώτο στοιχείο του, δηλαδή είναι ίσο με τη διεύθυνση του πρώτου στοιχείου του πίνακα

  Π.χ. αν έχουμε δηλώσει τον πίνακα `arr`, ισχύει ότι

  ```
  arr == &arr[0]
  ```

  και γενικά:

  ```
  arr + n == &arr[n]
  ```

- Επομένως, όταν μία συνάρτηση δέχεται σαν παράμετρο το όνομα ενός πίνακα, τότε στη συνάρτηση μεταβιβάζεται η διεύθυνση του πρώτου του στοιχείου και όχι ένα αντίγραφο του πίνακα

- Άρα, αφού η συνάρτηση έχει πρόσβαση στη διεύθυνση του πρώτου στοιχείου του πίνακα μπορεί να τροποποιήσει τις τιμές όλων των στοιχείων του πίνακα

Κλήση Συνάρτησης με παράμετρο Πίνακα (ΙΙΙ)

- Συνήθως, οι συναρτήσεις που δέχονται σαν παράμετρο έναν πίνακα δέχονται και **άλλη μία παράμετρο**, η οποία δηλώνει το **μέγεθος του πίνακα**

- Έτσι, το προηγούμενο παράδειγμα θα μπορούσε π.χ. να γίνει:

```
void function(int arr[],int size); /* Δήλωση συνάρτησης που έχει
παραμέτρους έναν πίνακα ακεραίων και μία ακέραια μεταβλητή και
δεν επιστρέφει τίποτα. */

int main(void)
{
	int pin[100];

	function(pin,100);/* Στην κλήση της συνάρτησης γράφουμε το
όνομα του πίνακα χωρίς τις αγκύλες και το μέγεθος του πίνακα. */
	return 0;
}

void function(int arr[],int size)
{
	/* Σώμα συνάρτησης. */
}
```

Παρατηρήσεις (I)

- Αφού το όνομα ενός πίνακα είναι **δείκτης στο πρώτο του στοιχείο**, τότε η δήλωση της συνάρτησης `function()` στο προηγούμενο παράδειγμα, καθώς και η επικεφαλίδα της, θα μπορούσε να γραφεί ισοδύναμα ως:

```
void function(int *arr);
```

- Ωστόσο, προτείνουμε τη σύνταξη:

```
void function(int arr[]);
```

έτσι ώστε να φαίνεται ξεκάθαρα ότι η συνάρτηση δέχεται σαν παράμετρο έναν πίνακα

Παρατηρήσεις (ΙΙ)

- Τα στοιχεία του πίνακα στο σώμα της συνάρτησης μπορούν να προσπελαστούν είτε με τον δείκτη θέσης του στοιχείου στον πίνακα είτε με σημειογραφία δείκτη

Π.χ.

```
void function(int arr[], int size)
{
        arr[0] = 10; /* Ισοδύναμο με *arr = 10; */
        arr[1] = 20; /* Ισοδύναμο με *(arr+1) = 20; */
}
```

Παρατηρήσεις (III)

- Αν θέλουμε μία συνάρτηση να ΜΗΝ μπορεί να αλλάζει τις τιμές των στοιχείων του πίνακα, τότε στη δήλωση της συνάρτησης χρησιμοποιούμε τη λέξη `const`

Π.χ. με την παρακάτω δήλωση η συνάρτηση `function()` δεν μπορεί να μεταβάλλει τις τιμές των στοιχείων του πίνακα `arr`

```
void function(const int arr[]);
```

Παραδείγματα (Ι)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>

void test(int arr[]);

int main(void)
{
        int i,array[5] = {10,20,30,40,50};

        test(array);
        for(i = 0; i < 5; i++)
                printf("%d ",array[i]);
        return 0;
}

void test(int arr[])
{
        arr[0] = arr[1] = 0;
}
```

Έξοδος: 0 0 30 40 50

Παραδείγματα (ΙΙ)

- Δημιουργήστε μία συνάρτηση που να δέχεται σαν παραμέτρους έναν πίνακα ακεραίων και το μέγεθός του, να εμφανίζει τα στοιχεία του πίνακα και να επιστρέφει τον μέσο όρο τους.

  Στη συνέχεια γράψτε ένα πρόγραμμα το οποίο να διαβάζει 5 ακεραίους, να τους αποθηκεύει σε έναν πίνακα ακεραίων και να εμφανίζει τον μέσο όρο των στοιχείων του πίνακα με χρήση της συνάρτησης

```c
#include <stdio.h>

float avg_arr(int arr[],int size);

int main(void)
{
	int i,arr[5];

	for(i = 0; i < 5; i++)
	{
		printf("Enter number: ");
		scanf("%d",&arr[i]);
	}
	printf("Avg = %.2f\n",avg_arr(arr,5));
	return 0;
}

float avg_arr(int arr[],int size)
{
	int i,sum;

	sum = 0;
	for(i = 0; i < size; i++)
	{
		printf("%d ",arr[i]);
		sum += arr[i];
	}
	return (float)sum/size;
}
```

Παραδείγματα (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>

void test(int* ptr1,int* ptr2);

int main(void)
{
        int i = 10,j = 20;

        test(&i,&j);
        printf("i = %d,j = %d\n",i,j);
        return 0;
}

void test(int* ptr1,int* ptr2)
{
        ptr1 = ptr2;
        *ptr1 = 100;
}
```

Έξοδος: `i = 10, j = 100`

Παραδείγματα (IV)

- Δημιουργήστε μία συνάρτηση που να δέχεται σαν παραμέτρους δύο δείκτες σε double και να επιστρέφει έναν δείκτη στον double που έχει τη μεγαλύτερη τιμή

Στη συνέχεια γράψτε ένα πρόγραμμα το οποίο να διαβάζει δύο double και να εμφανίζει τον μεγαλύτερο από αυτούς με χρήση της συνάρτησης

```
#include <stdio.h>

double *max(double *ptr1, double *ptr2);

int main(void)
{
	double *ptr, i, j;

	printf("Enter numbers: ");
	scanf("%lf%lf", &i, &j);

	ptr = max(&i, &j);
	printf("The max of %f and %f is %f\n", i, j, *ptr);
	return 0;
}

double *max(double *ptr1, double *ptr2)
{
	if(*ptr1 > *ptr2)
		return ptr1;
	else
		return ptr2;
}
```

Δήλωση Συνάρτησης με παράμετρο Διδιάστατο Πίνακα

- Για τη **δήλωση** μίας συνάρτησης που να έχει σαν παράμετρο έναν διδιάστατο πίνακα, απαιτείται να γράψουμε **το όνομα** του πίνακα ακολουθούμενο από τις **μέγιστες τιμές** των διαστάσεών του

- Π.χ. η δήλωση μίας συνάρτησης με όνομα `test` η οποία θα δέχεται σαν παράμετρο έναν διδιάστατο πίνακα ακεραίων διαστάσεων `5×10` θα μπορούσε να ήταν:

```
void test(int arr[5][10]);
```

- Εναλλακτικά, κατά τη δήλωση είναι δυνατόν **να παραλειφθεί** η τιμή της **πρώτης** διάστασης

- Δηλαδή, μπορούμε να γράψουμε:

```
void test(int arr[][10]);
```

Παρατηρήσεις (1/3)

- Αφού, όπως έχουμε ήδη πει, η C χειρίζεται έναν διδιάστατο πίνακα σαν πίνακα από μονοδιάστατους πίνακες, ο μεταγλωττιστής στην πραγματικότητα μετατρέπει τον διδιάστατο πίνακα της παραμέτρου ως «δείκτη σε πίνακα»

- Επομένως, θα μπορούσαμε ισοδύναμα αντί για:

```
void test(int arr[][10]);
```

να γράψουμε:

```
void test(int (*arr)[10]);
```

- Δεδομένου ότι οι αγκύλες `[]` έχουν μεγαλύτερη προτεραιότητα από το `*`, οι παρενθέσεις είναι απαραίτητες (αλλιώς, ο `arr` θα μεταφραστεί ως πίνακας `10` δεικτών σε ακεραίους αντί για δείκτης σε πίνακα `10` ακεραίων)

- Παρόλο που και η δεύτερη δήλωση είναι σωστή, προτιμούμε την πρώτη, που δείχνει ξεκάθαρα ότι η παράμετρος είναι ένας διδιάστατος πίνακας

Παρατηρήσεις (2/3)

Για να δούμε τι θυμάστε για έναν διδιάστατο πίνακα, έστω `arr[10][20]`...

- Πού δείχνει ο `arr`?
  - Στο πρώτο στοιχείο της πρώτης γραμμής του διδιάστατου πίνακα
- Και πού θα δείχνει αν γράψουμε `arr++`?
  - Στο επόμενο στοιχείο, δηλαδή στο πρώτο στοιχείο της δεύτερης γραμμής του πίνακα (αυξάνεται κατά το μέγεθος της γραμμής)
- Και τι είναι το `(*arr)[1]`?
  - Είναι το δεύτερο στοιχείο της γραμμής στην οποία δείχνει ο `arr`
- Και πώς ξέρουμε ότι φτάσαμε στην τελευταία γραμμή του πίνακα?
  - Συνήθως, είτε χρησιμοποιούμε κάποια σταθερά (π.χ., `#define ROWS 20`) ή περνάμε κάποιο όρισμα που δηλώνει τον συνολικό αριθμό των γραμμών

- Ας το γράψουμε άλλη μια φορά: Ο μεταγλωττιστής μεταφράζει τις **παραμέτρους διδιάστατους πίνακες** μιας συνάρτησης **ως δείκτες σε μονοδιάστατο πίνακα** (μεγέθους όσες οι στήλες του διδιάστατου) και όχι ως δείκτη σε δείκτη που ίσως νομίζετε
- Επομένως, απαγορεύεται να γράψετε: `void test(int **arr);`

Παρατηρήσεις (3/3)

- Όταν θα δούμε τη συνάρτηση `malloc()` στο Κεφάλαιο 14, θα μάθουμε πώς μπορούμε να χρησιμοποιήσουμε έναν δείκτη σε δείκτη για να δημιουργήσουμε δυναμικά έναν διδιάστατο πίνακα και να τον χρησιμοποιήσουμε σε μία συνάρτηση

- Για την ώρα, ας δούμε συγκεντρωτικά πώς μεταφράζει ο μεταγλωττιστής τις παραμέτρους πίνακες:

Μονοδιάστατος: `arr[10]` **μεταφράζεται σε δείκτη** `*arr`

Διδιάστατος: `arr[][20]` **μεταφράζεται σε δείκτη σε πίνακα** `(*arr)[20]`

Πίνακας δεικτών: `*arr[50]` **μεταφράζεται σε δείκτη σε δείκτη** `**arr`

Κλήση Συνάρτησης με παράμετρο Διδιάστατο Πίνακα

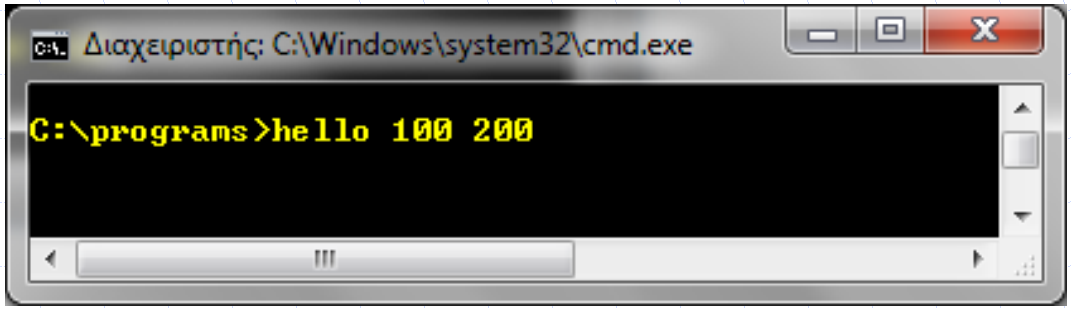
- Όπως και στην περίπτωση κλήσης συνάρτησης που έχει ως παράμετρο μονοδιάστατο πίνακα, έτσι και κατά την **κλήση** μίας συνάρτησης που έχει σαν παράμετρο έναν διδιάστατο πίνακα, γράφουμε **μόνο** το **όνομα** του πίνακα, **χωρίς τις αγκύλες**, όπως φαίνεται στο παρακάτω παράδειγμα:

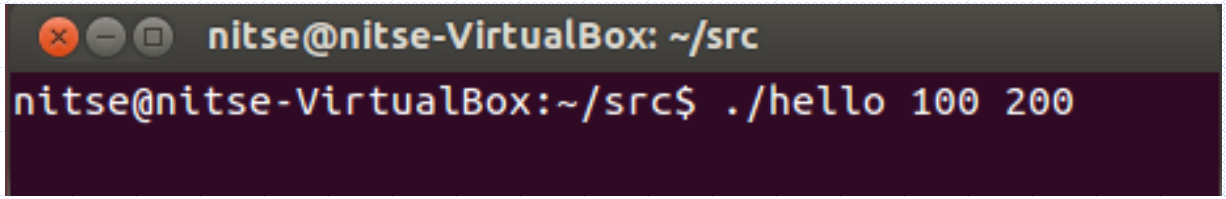
```
void function(int arr[5][10]); /* Δήλωση συνάρτησης που έχει σαν
παράμετρο έναν διδιάστατο πίνακα ακεραίων και δεν επιστρέφει
τίποτα. */

int main(void)
{
	int pin[5][10];

	function(pin); /* Στην κλήση της συνάρτησης γράφουμε το
όνομα του πίνακα χωρίς τις αγκύλες. */
	return 0;
}

void function(int arr[5][10])
{
	/* Σώμα συνάρτησης. */
}
```

Διοχέτευση Πληροφορίας στη Συνάρτηση `main()` (Ι)

- Για τη διοχέτευση πληροφορίας σε ένα πρόγραμμα τη στιγμή της **εκκίνησής** του, γράφουμε στη γραμμή εντολών (command line) το όνομα του εκτελέσιμου αρχείου και τις επιθυμητές τιμές

- Π.χ. αν ο μεταγλωττιστής έχει δημιουργήσει το εκτελέσιμο αρχείο `hello.exe` στον φάκελο `C:\programs` σε περιβάλλον Windows (ή `hello` σε Linux/Unix στον φάκελο `src`), τότε, γράφοντας στη γραμμή εντολών:





Θα εκτελεστεί το πρόγραμμα `hello` και στη συνάρτηση `main()` του προγράμματος θα διοχετευθούν οι τιμές `100` και `200`

Διοχέτευση Παραμέτρων στη Συνάρτηση `main()` (II)

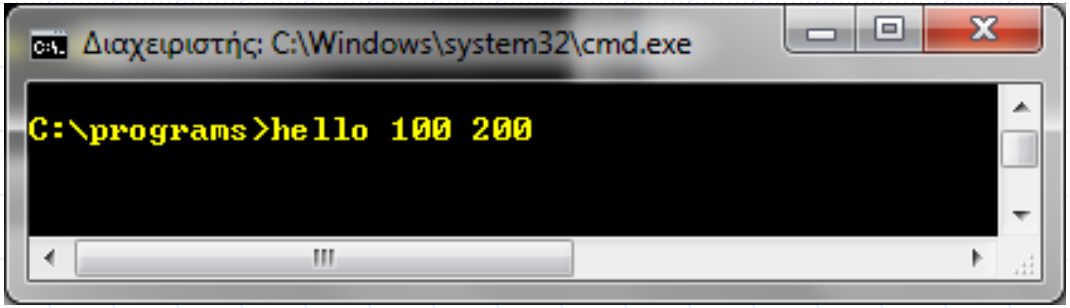
- Όμως, για να είναι σε θέση η συνάρτηση `main()` ενός προγράμματος να διαβάσει τις τιμές των παραμέτρων από τη γραμμή εντολών **θα πρέπει** να έχει δηλωθεί ως:

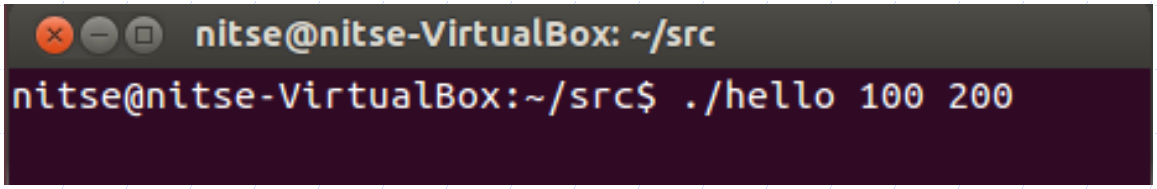
```
int main(int argc, char *argv[])
```

α) Η παράμετρος `argc` είναι **ένας ακέραιος αριθμός** που δηλώνει **το πλήθος των ορισμάτων** στη γραμμή εντολών

Η τιμή του `argc` είναι τουλάχιστον `1`, γιατί το όνομα του προγράμματος θεωρείται κι αυτό ένα όρισμα

Π.χ. στη γραμμή εντολών:
η τιμή του `argc` είναι
ίση με `3`





Διοχέτευση Παραμέτρων στη Συνάρτηση `main()` (III)

β) Η παράμετρος `argv` είναι ένας πίνακας δεικτών προς τα ορίσματα της γραμμής εντολών

Τα ορίσματα της γραμμής εντολών αποθηκεύονται ως αλφαριθμητικά, γι' αυτό και η παράμετρος `argv` δηλώνεται σαν ένας πίνακας δεικτών προς χαρακτήρα

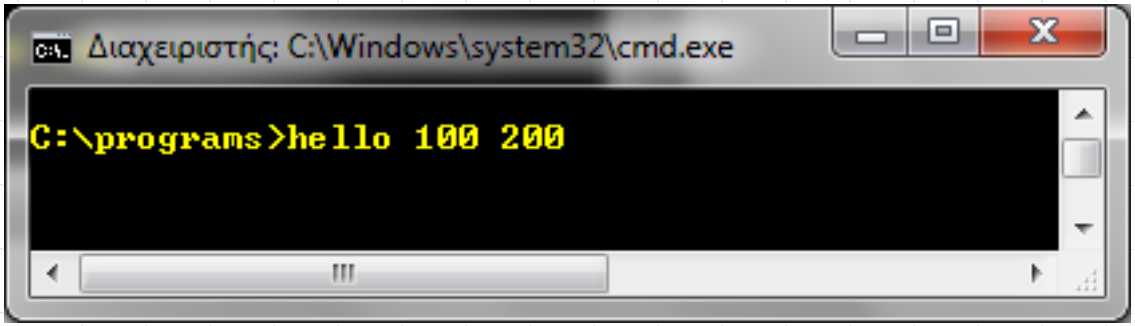
Οι δείκτες είναι αποθηκευμένοι στον πίνακα `argv` από τη θέση `0` έως και τη θέση `argc-1`

Συγκεκριμένα, ο δείκτης `argv[0]` δείχνει στο πρώτο όρισμα της γραμμής εντολών που είναι το όνομα του προγράμματος, ενώ οι υπόλοιποι δείκτες δείχνουν στα επόμενα ορίσματα (δηλ. ο `argv[1]` δείχνει στο δεύτερο όρισμα, ο `argv[2]` δείχνει στο τρίτο όρισμα, ... και ο `argv[argc-1]` δείχνει στο τελευταίο όρισμα της γραμμής εντολών)

Διοχέτευση Παραμέτρων στη Συνάρτηση `main()` (IV)

Ο πίνακας `argv` έχει ένα ακόμα στοιχείο, το `argv[argc]`, του οποίου η τιμή είναι ίση με `NULL`

Π.χ. για την επόμενη γραμμή εντολών:



όπως εξηγήθηκε προηγουμένως το `argc` είναι ίσο με 3 ενώ οι παράμετροι της γραμμής εντολών `hello`, `100` και `200` **θεωρούνται όλες αλφαριθμητικά**, και έτσι:

- ο δείκτης `argv[0]` δείχνει στο αλφαριθμητικό `"hello"`
- ο δείκτης `argv[1]` δείχνει στο αλφαριθμητικό `"100"`
- ο δείκτης `argv[2]` δείχνει στο αλφαριθμητικό `"200"` και τέλος
- ο δείκτης `argv[3]` είναι ίσος με `NULL`

Παρατηρήσεις

- Η διοχέτευση πληροφορίας σε ένα πρόγραμμα, μέσω της γραμμής εντολών, είναι ένας **εναλλακτικός τρόπος** εισαγωγής δεδομένων στο πρόγραμμα
- Προφανώς, αυτή η πληροφορία μπορεί να διαβαστεί με τη χρήση εντολών ανάγνωσης δεδομένων, όπως η `scanf()` και η `gets()`

- Οι παράμετροι στη γραμμή εντολών **πρέπει να διαχωρίζονται** μεταξύ τους με **κενό** διάστημα *(space)*

- Αντί των ονομάτων `argc` και `argv` στη συνάρτηση `main()` μπορείτε να χρησιμοποιήσετε όποια ονόματα επιθυμείτε
- Ωστόσο, συνηθίζεται από τους προγραμματιστές να χρησιμοποιούν αυτά τα ονόματα και όχι κάποια άλλα (όχι μόνο στη C αλλά και σε άλλες γλώσσες προγραμματισμού, για τη διοχέτευση παραμέτρων στη συνάρτηση `main()`)

Παράδειγμα

- Τι κάνει το παρακάτω πρόγραμμα ???

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

int main(int argc,char* argv[])
{
    if(argc == 1)
        printf("Error: missing user name and password\n");
    else if(argc == 2)
        printf("Error: missing password\n");
    else if(argc == 3)
    {
        if(strcmp(argv[1],"user") == 0 &&
            strcmp(argv[2],"pswd") == 0)
            printf("Valid user. The program ""%s"" will be
executed ...\n",argv[0]);
        else
            printf("Wrong data\n");
    }
    else
        printf("Error: too many parameters\n");

    return 0;
}
```

Αν δεν είναι συνολικά τρεις, τότε τυπώνει στην οθόνη κατάλληλο μήνυμα

Αν είναι τρεις, τότε ελέγχει αν οι τιμές της 2ης και της 3ης παραμέτρου είναι ίσες με κάποιες συγκεκριμένες τιμές και τυπώνει στην οθόνη αντίστοιχο μήνυμα

Ελέγχει τις παραμέτρους της γραμμής εντολών και...

Συναρτήσεις με μεταβλητό αριθμό παραμέτρων

- Μία συνάρτηση είναι δυνατόν να δέχεται **μεταβλητό** αριθμό παραμέτρων

- Για να δηλώσουμε μία τέτοια συνάρτηση γράφουμε **αρχικά** τις **σταθερές της παραμέτρους**, δηλαδή αυτές που θα υπάρχουν **πάντα** και μετά προσθέτουμε **αποσιωπητικά** (...).

- Π.χ. η συνάρτηση:

```
void test(int num, char *str, ...);
```

είναι μία συνάρτηση που δέχεται δύο σταθερές παραμέτρους (μία ακέραια μεταβλητή και έναν δείκτη σε χαρακτήρα) και στη συνέχεια έναν μεταβλητό αριθμό παραμέτρων

Κλήση Συνάρτησης με μεταβλητό αριθμό παραμέτρων

- Η κλήση μίας τέτοιας συνάρτησης γίνεται γράφοντας το όνομα της συνάρτησης, τις τιμές των **σταθερών** παραμέτρων και τις τιμές των **μη-σταθερών** παραμέτρων

- Παράδειγμα κλήσης της προηγούμενης συνάρτησης που είχε τη δήλωση:

```
void test(int num, char *str, ...);
```

Θα μπορούσε να ήταν:

```
test(3, "keimeno", 5, 8.9, "sample");
```

- **Οι σταθερές παράμετροι** αυτής έχουν τιμές `3` και `"keimeno"` αντίστοιχα
- Οι τύποι δεδομένων **των μη-σταθερών παραμέτρων** αυτής της συνάρτησης είναι `int`, `float` και `char*` με τιμές `5`, `8.9` και `"sample"`, αντίστοιχα

Παρατηρήσεις

- Μία συνάρτηση που δέχεται έναν μεταβλητό αριθμό παραμέτρων πρέπει να έχει τουλάχιστον μία σταθερή παράμετρο
- Οι περιπτώσεις που θα χρειαστείτε να δηλώσετε συναρτήσεις με μεταβλητό αριθμό παραμέτρων θα είναι από ελάχιστες έως μηδενικές...
- Ωστόσο, αν ανοίξετε το αρχείο `stdio.h` και δείτε τις δηλώσεις (τα πρωτότυπα) των δύο πιο συνηθισμένων συναρτήσεων, της `printf()` και της `scanf()`, θα δείτε ότι δηλώνονται σαν συναρτήσεις που δέχονται μεταβλητό αριθμό παραμέτρων

Αναδρομικές Συναρτήσεις (Ι)

- Μία συνάρτηση μπορεί να καλεί μέσα στο σώμα της οποιαδήποτε άλλη συνάρτηση, ακόμα και τον εαυτό της
- Μία συνάρτηση που μέσα στο σώμα της **καλεί τον εαυτό της** ονομάζεται **αναδρομική συνάρτηση**
- Για να εξηγήσουμε πως λειτουργεί μία αναδρομική συνάρτηση θα εξετάσουμε τη συνάρτηση `show()` του επόμενου παραδείγματος
- Η συνάρτηση `show()` βλέπουμε ότι είναι αναδρομική, αφού μέσα στο σώμα της καλεί τον εαυτό της

Αναδρομικές Συναρτήσεις (ΙΙ)

```
#include <stdio.h>

void show(int num);

int main(void)
{
        int i;

        printf("Enter number: ");
        scanf("%d", &i);

        show(i);
        return 0;
}

void show(int num)
{
        if(num > 1)
                show(num-1);

        printf("val = %d\n", num);
}
```

Αν ο χρήστης πληκτρολογήσει 3:

Έξοδος: `val = 1`
`val = 2`
`val = 3`

Διότι, όταν μία συνάρτηση καλεί τον εαυτό της, οι επόμενες εντολές του σώματός της καθώς και οι τιμές των εμπλεκόμενων μεταβλητών (οι οποίες αποθηκεύονται) παραμένουν στη μνήμη.

Όταν η συνάρτηση σταματήσει να καλεί τον εαυτό της, οι αποθηκευμένες εντολές εκτελούνται με αντίστροφη σειρά (δηλ. από την τελευταία προς την πρώτη)

Παρατηρήσεις (Ι)

- Οι εντολές και οι μεταβλητές που υπάρχουν μετά την κλήση μίας αναδρομικής συνάρτησης αποθηκεύονται σε ένα ειδικό τμήμα της μνήμης (στοίβα - stack) και **θα εκτελεστούν μόνο όταν η συνάρτηση δεν καλέσει πάλι τον εαυτό της**

- Λάβετε όμως υπ' όψιν ότι το μέγεθος της στοίβας δεν είναι ιδιαίτερα μεγάλο, το οποίο σημαίνει ότι δεν μπορεί να αποθηκευτεί μεγάλος όγκος εντολών και μεταβλητών

- Π.χ. αν στο προηγούμενο πρόγραμμα ο χρήστης εισάγει την τιμή 50000 ή μία μεγαλύτερη τιμή είναι πολύ πιθανό το πρόγραμμα να μην εκτελεστεί και ο μεταγλωττιστής να εμφανίσει το μήνυμα: `stack overflow`

- Αυτό το μήνυμα σημαίνει ότι δεν υπάρχει διαθέσιμος χώρος στη στοίβα για να αποθηκευτεί η κάθε μεταβλητή `num` του προηγούμενου παραδείγματος και η πληροφορία για την αντίστοιχη `printf()`

- Άρα, **η πιθανότητα δέσμευσης όλης της διαθέσιμης μνήμης** είναι **ο πρώτος λόγος** για τον οποίον **δεν προτείνεται** η χρήση αναδρομικών συναρτήσεων (αν μπορεί το πρόβλημα να επιλυθεί με εναλλακτικό τρόπο, π.χ. με χρήση επαναληπτικού βρόχου)

Παρατηρήσεις (ΙΙ)

- Όταν μία αναδρομική συνάρτηση καλεί πολλές φορές τον εαυτό της, τότε **ο χρόνος εκτέλεσης** της συνάρτησης μπορεί να γίνει **υπερβολικά μεγάλος**

- Αυτός είναι **ο δεύτερος λόγος** για τον οποίο **δεν προτείνεται** η χρήση αναδρομικών συναρτήσεων – αν φυσικά μπορεί να αποφευχθεί

Μία αναδρομική συνάρτηση πρέπει να περιέχει μία συνθήκη τερματισμού, αλλιώς θα εκτελείται συνεχώς

- Στο προηγούμενο παράδειγμα, αυτή η συνθήκη ήταν η εντολή:

```
if(num > 1)
```

Παράδειγμα (Ι)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει έναν ακέραιο αριθμό (n) και να εμφανίζει το παραγοντικό του (n!) με χρήση αντίστοιχης αναδρομικής συνάρτησης n!=n*(n-1)!

```
#include <stdio.h>

double fact(int num);

int main(void)
{
	int num;

	do
	{
		printf("Enter a positive integer less than 170: ");
		scanf("%d", &num);
	} while(num < 0 || num > 170);

	printf("Factorial of %d is %e\n", num, fact(num));
	return 0;
}

double fact(int num)
{
	if((num == 0) || (num == 1))
		return 1;
	else
		return num * fact(num-1);
}
```

Παράδειγμα (ΙΙ)

- Δημιουργήστε μία αναδρομική συνάρτηση που να δέχεται σαν παράμετρο έναν θετικό ακέραιο, να τον εμφανίζει στην οθόνη και να υπολογίζει τον επόμενο ακέραιο βάσει της «Εικασίας του Collatz».

  Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν θετικό ακέραιο και να εμφανίζει τους αριθμούς που προκύπτουν βάσει της «Εικασίας του Collatz» με χρήση της συνάρτησης

- Ποια είναι η «Εικασία του Collatz» ???

  Σκεφτείτε έναν θετικό ακέραιο αριθμό ν και εκτελέστε τον εξής αλγόριθμο:

  - Αν είναι άρτιος, υποδιπλασιάστε τον (ν/2)
  - Αν είναι περιττός, τριπλασιάστε τον και προσθέστε του τη μονάδα (3ν+1)

  Επαναλάβετε την παραπάνω διαδικασία για τον αριθμό που προκύπτει και, τότε, θα παρατηρήσετε το εξής εκπληκτικό: όποιον θετικό ακέραιο και να είχατε αρχικά επιλέξει, πάντοτε θα καταλήγετε στο…1!!!

  Π.χ. αν είχατε επιλέξει τον αριθμό 53, τότε:

  53 -> 160 -> 80 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1 !!!

Λύση Παραδείγματος (ΙΙ)

```c
#include <stdio.h>

int collatz(int n);

int main(void)
{
	int a;

	do
	{
		printf("Enter a positive integer: ");
		scanf("%d", &a);
	} while(a <= 0);

	printf("The result is %d !!!\n", collatz(a));
	return 0;
}
int collatz(int n)
{
	printf("%d\n", n);

	if(n == 1)
		return 1;
	else if(n & 1) /* Αν ο n είναι περιττός. */
		return collatz(3*n+1);
	else /* Αν ο n είναι άρτιος. */
		return collatz(n/2);
}
```

Συναρτήσεις Βιβλιοθήκης

- Η C παρέχει έτοιμες υλοποιημένες συναρτήσεις, μέσω των header files που ανήκουν στις αντίστοιχες βιβλιοθήκες της, οι οποίες εκτελούν συγκεκριμένες εργασίες
- Τα πλέον χρησιμοποιούμενα header files της C παρουσιάζονται στον παρακάτω πίνακα (τις συναρτήσεις όλων των header files της C μπορείτε να τις βρείτε στο Παράρτημα Γ του βιβλίου)

Header file	Επεξήγηση
`<ctype.h>`	Περιέχει πρωτότυπα συναρτήσεων που ελέγχουν ορισμένες ιδιότητες χαρακτήρων, καθώς και πρωτότυπα συναρτήσεων για συναρτήσεις που μπορούν να χρησιμοποιηθούν για την μετατροπή πεζών (μικρών) γραμμάτων σε κεφαλαία γράμματα και αντίστροφα.
`<math.h>`	Περιέχει πρωτότυπα συναρτήσεων για μαθηματικές συναρτήσεις βιβλιοθήκης.
`<stdio.h>`	Περιέχει πρωτότυπα συναρτήσεων για standard συναρτήσεις εισόδου / εξόδου καθώς και πληροφορίες που χρησιμοποιούνται από αυτές.
`<stdlib.h>`	Περιέχει πρωτότυπα συναρτήσεων για μετατροπή αριθμών σε κείμενο και κειμένου σε αριθμούς, κατανομή μνήμης (memory allocation), τυχαίους αριθμούς και άλλες γενικής χρήσης συναρτήσεις.
`<string.h>`	Περιέχει πρωτότυπα συναρτήσεων για την επεξεργασία αλφαριθμητικών.
`<time.h>`	Περιέχει πρωτότυπα συναρτήσεων και τύπους για το χειρισμό ώρας και ημερομηνίας.

ctype.h (I)

- Περιλαμβάνει συναρτήσεις για την εκτέλεση χρήσιμων ελέγχων και χειρισμών δεδομένων τύπου χαρακτήρα (char)

Πρωτότυπο συνάρτησης	Περιγραφή
`int isdigit( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής ch αντιστοιχεί σε έναν από τους χαρακτήρες 0-9, αλλιώς επιστρέφει 0
`int isalpha( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής ch αντιστοιχεί σε έναν από τους χαρακτήρες a-z ή A-Z, αλλιώς επιστρέφει 0
`int isalnum( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής ch αντιστοιχεί σε έναν από τους χαρακτήρες a-z, A-Z ή 0-9, αλλιώς επιστρέφει 0
`int isxdigit( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής ch αντιστοιχεί σε έναν από τους χαρακτήρες που χρησιμοποιούνται στο δεκαεξαδικό σύστημα, δηλαδή a-f, A-F ή 0-9, αλλιώς επιστρέφει 0
`int islower( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής ch αντιστοιχεί σε έναν από τους πεζούς χαρακτήρες a-z, αλλιώς επιστρέφει 0
`int isupper( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής ch αντιστοιχεί σε έναν από τους κεφαλαίους χαρακτήρες A-Z, αλλιώς επιστρέφει 0

ctype.h (II)

Πρωτότυπο συνάρτησης	Περιγραφή
`int isspace( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής `ch` αντιστοιχεί σε έναν χαρακτήρα διαστήματος, αλλιώς επιστρέφει 0
`int iscntrl( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής `ch` αντιστοιχεί σε έναν χαρακτήρα ελέγχου, αλλιώς επιστρέφει 0
`int ispunct( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής `ch` αντιστοιχεί σε έναν χαρακτήρα που να είναι σημείο στίξης, αλλιώς επιστρέφει 0
`int isprint( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής `ch` αντιστοιχεί σε έναν εκτυπώσιμο χαρακτήρα, αλλιώς επιστρέφει 0
`int isgraph( int ch );`	Επιστρέφει μία μη μηδενική τιμή, αν η τιμή της μεταβλητής `ch` αντιστοιχεί σε έναν εκτυπώσιμο χαρακτήρα εκτός του «κενού διαστήματος», αλλιώς επιστρέφει 0
`int tolower( int ch );`	Επιστρέφει τον πεζό χαρακτήρα που αντιστοιχεί στην τιμή της μεταβλητής `ch`
`int toupper( int ch );`	Επιστρέφει τον κεφαλαίο χαρακτήρα που αντιστοιχεί στην τιμή της μεταβλητής `ch`

math.h

Συνάρτηση	Περιγραφή	Παράδειγμα
`sqrt( x )`	Τετραγωνική ρίζα του *x*	`sqrt( 900.0 ) "επιστρέφει" 30.0` `sqrt( 9.0 ) "επιστρέφει" 3.0`
`exp( x )`	Εκθετική συνάρτηση e^x	`exp( 1.0 ) "επιστρέφει" 2.718282` `exp( 2.0 ) "επιστρέφει" 7.389056`
`log( x )`	Λογάριθμος του *x* με βάση το *e* (*ln*x)	`log( 2.718282 ) "επιστρέφει" 1.0` `log( 7.389056 ) "επιστρέφει" 2.0`

`log10( x )`	Λογάριθμος του *x* (με βάση το 10)	`log10( 1.0 ) "επιστρέφει" 0.0`
		`log10( 10.0 ) "επιστρέφει" 1.0`
		`log10( 100.0 ) "επιστρέφει" 2.0`

`fabs( x )`	Απόλυτη τιμή του *x*	`fabs( 5.0 ) "επιστρέφει" 5.0`
		`fabs( 0.0 ) "επιστρέφει" 0.0`
		`fabs( -5.0 ) "επιστρέφει" 5.0`

`floor( x )`	Στρογγυλοποιεί το *x* στον μεγαλύτερο ακέραιο (μικρότερο ή ίσο του *x*)	`floor( 9.2 ) "επιστρέφει" 9.0` `floor( -9.8 ) "επιστρέφει" -10.0`
`ceil( x )`	Στρογγυλοποιεί το *x* στον μικρότερο ακέραιο (μεγαλύτερο ή ίσο του *x*)	`ceil( 9.2 ) "επιστρέφει" 10.0` `ceil( -9.8 ) "επιστρέφει" -9.0`
`pow( x, y )`	*x* εις την *y* (x^y)	`pow( 2, 7 ) "επιστρέφει" 128.0` `pow( 9, .5 ) "επιστρέφει" 3.0`
`fmod( x, y )`	υπόλοιπο της διαίρεσης *x*/*y* σαν αριθμό κινητής υποδιαστολής	`fmod( 13.657, 2.333 ) "επιστρέφει" 1.992`
`sin( x )`	ημίτονο του *x* (*x* σε ακτίνια)	`sin( 0.0 ) "επιστρέφει" 0.0`
`cos( x )`	συνημίτονο του *x* (*x* σε ακτίνια)	`cos( 0.0 ) "επιστρέφει" 1.0`
`tan( x )`	εφαπτομένη του *x* (*x* σε ακτίνια)	`tan( 0.0 ) "επιστρέφει" 0.0`

Παρατηρήσεις

- Οι σημαντικότερες συναρτήσεις βιβλιοθήκης από τα header files `stdlib.h` και `string.h` θα παρουσιαστούν στην ενότητα των αλφαριθμητικών

- Περισσότερες λεπτομέρειες για όλες τις συναρτήσεις βιβλιοθήκης της C μπορείτε να βρείτε στο Παράρτημα Γ' του βιβλίου

Δημιουργία Τυχαίων Αριθμών (Ι)

- Μία χρήσιμη συνάρτηση του `stdlib.h` είναι η συνάρτηση `rand()`, η οποία χρησιμοποιείται για τη δημιουργία τυχαίων αριθμών

  `int rand()` : Επιστρέφει έναν τυχαίο θετικό ακέραιο αριθμό από 0 έως `RAND_MAX` (η τιμή της `RAND_MAX` δηλώνεται στο αρχείο `stdlib.h` και είναι ίση με 32767)

- Για τη δημιουργία ενός τυχαίου ακέραιου μεταξύ 0 και n-1?

```
rand()%n
```

- Για τη δημιουργία ενός τυχαίου ακέραιου μεταξύ 1 και n?

```
1 + rand()%n
```

- Π.χ. εξομοίωση ρίψης ζαριού (αποτέλεσμα από 1 έως και 6)???

```
1 + rand()%6
```

Δημιουργία Τυχαίων Αριθμών (ΙΙ)

- Μία επίσης χρήσιμη συνάρτηση του `stdlib.h` είναι η συνάρτηση `srand()`, η οποία χρησιμοποιείται για την «αρχικοποίηση» της γεννήτριας παραγωγής τυχαίων αριθμών

```
void srand(unsigned int seed)
```

- Αφού χρησιμοποιήσουμε μία φορά στο πρόγραμμά μας τη συνάρτηση `srand()`, στη συνέχεια χρησιμοποιούμε τη συνάρτηση `rand()` (όπως δείξαμε προηγουμένως) για τη δημιουργία των τυχαίων αριθμών

- Το όρισμα `seed` της συνάρτησης `srand()` χρησιμοποιείται για να ορίσει την «τυχαιότητα» των επιστρεφόμενων τιμών της `rand()`

- Συνήθως, η τιμή που χρησιμοποιείται σαν `seed` είναι η επιστροφή της συνάρτησης `time()` (η οποία περιέχεται στο `time.h`), ώστε κάθε φορά που εκτελείται το πρόγραμμα η `rand()` να επιστρέφει διαφορετικούς αριθμούς

- Η συνάρτηση `time()` με όρισμα `NULL` επιστρέφει τον αριθμό των δευτερολέπτων που έχουν περάσει τη στιγμή που καλείται η συνάρτηση από τη χρονική στιγμή (00:00:00),January 1,1970

Παράδειγμα

- Γράψτε ένα πρόγραμμα το οποίο να εξομοιώνει 10 συνεχόμενες τυχαίες ρίψεις ενός ζαριού και να εκτυπωνει το αποτέλεσμα της κάθε ρίψης στην οθόνη

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

int main(void)
{
    int i,dice;

/* "randomize" the random number generator by using current time */
    srand(time(NULL));

    for (i=0;i<10;i++)
    {
        dice = 1 + rand()%6;
        printf("The dice is now: %d \n",dice);
    }
    return 0;
}
```

Θα λειτουργήσει το πρόγραμμα αν παραλείψετε αυτή τη γραμμή???