C: functions basics

Vasilios Papaliakos, < >

Hit the space bar or swipe left for next slide

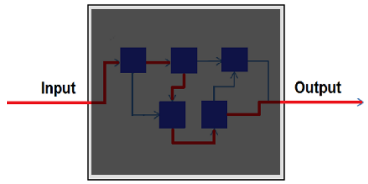
Σκοποί μαθήματος

Στο τέλος του μαθήματος, θα μπορείτε:

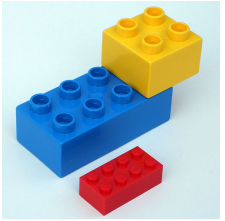
- να εξηγείτε την έννοια του αρθρωτού προγραμματισμού (modular programming)
- να ονοματίζετε τα 4 σημαντικά στοιχεία κάθε συνάρτησης
- να περιγράφετε τις 3 διαφορετικές ενέργειες που πρέπει να γίνουν για κάθε συνάρτηση (δήλωση, ορισμός, κλήση)
- να δημιουργείτε συναρτήσεις και να τις χρησιμοποιείτε
- να εξηγείτε γιατί συμπεριλαμβάνουμε τις επικεφαλίδες των βιβλιοθηκών στα προγράμματα

Αρθρωτός προγραμματισμός (modular programming)

- Άρθρωμα (module): κομμάτι προγράμματος με συγκεκριμένο ρόλο / λειτουργία
- Χαρακτηριστικά:
  - *κάθε άρθρωμα επιτελεί μια μόνο, σαφώς καθορισμένη λειτουργία*
  - *έτσι, κάθε άρθρωμα είναι "ανεξάρτητο" από τα υπόλοιπα*
  - *Black box concept: δεν (χρειάζεται να) γνωρίζουμε τον τρόπο υλοποίησης (**πως**), μόνο τη λειτουργικότητα (**τι** κάνει και πως θα το χρησιμοποιήσουμε)*
- Πλεονεκτήματα:
  - *διαχωρισμός λειτουργιών = διαχωρισμός προβλημάτων (separation of concerns)*
  - *ανεξάρτητο άρθρωμα = εναλλάξιμο (module replacement)*
  - *δυνατότητα επαναχρησιμοποίησης κώδικα (code reuse)*
  - *απλοποίηση κώδικα*
  - *ευκολότερη διαχείριση (πχ. εντοπισμός / διόρθωση σφαλμάτων)*

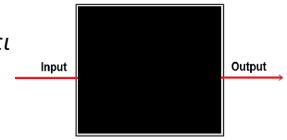


Giladgar [CC BY-SA 3.0], via Wikimedia Commons

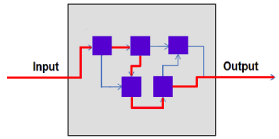


Black box / white box concept:

- *Black box concept: Δεν μπορούμε να δούμε μέσα στο "κουτί", δηλαδή δεν γνωρίζουμε **ΠΩΣ** υλοποιεί τις λειτουργίες του εσωτερικά. Βλέπουμε και γνωρίζουμε όμως **ΤΙ** κάνει, δηλαδή ποιες λειτουργίες υποστηρίζει (inputs / outputs).*

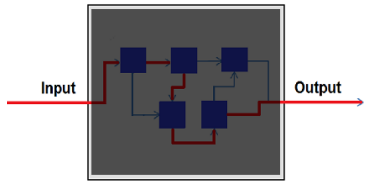


- *White box concept: Μπορούμε να δούμε και μέσα στο "κουτί", δηλαδή τον τρόπο (**ΠΩΣ**). Ταυτόχρονα, μπορούμε να δούμε και εξωτερικά του κουτιού, όπως κ στο black box.*

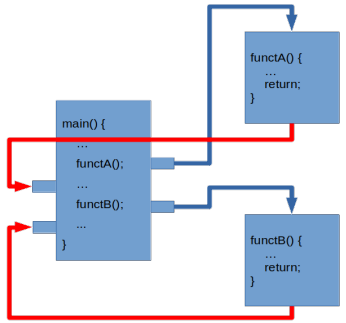


Συνάρτηση στη C

- Συνάρτηση (στη C) είναι μια αυτόνομη μονάδα κώδικα που εκτελεί μια συγκεκριμένη εργασία
- Μια συνάρτηση μπορεί να είναι ένα "μαύρο κουτί", στο οποίο δίνουμε πληροφορίες και μας επιστρέφει απαντήσεις
- Μέχρι στιγμής έχουμε δει στον κώδικά μας μόνο τη (βασική) συνάρτηση main(), από την οποία ξεκινάει η εκτέλεση του κώδικα
  - *ή μήπως όχι;*
  - *μήπως έχουμε δει κι άλλες (έτοιμες) συναρτήσεις;*
- Έλεγχος ροής προγράμματος:
  - *όταν καλείται μια συνάρτηση, ο έλεγχος ροής μεταβιβάζεται σε αυτή*
  - *στο τέλος κάθε συνάρτησης, ο έλεγχος επιστρέφει (return) στο σημείο του κώδικα που την κάλεσε*



Giladgar [CC BY-SA 3.0], via Wikimedia Commons



Συνάρτηση: 4 σημαντικά στοιχεία & 3 διαφορετικές ενέργειες

- Κάθε συνάρτηση έχει 4 σημαντικά στοιχεία:
  - *ένα όνομα*
  - *(συνήθως) επιστρέφουν μια τιμή συγκεκριμένου τύπου (πχ. int)*
  - *μπορεί να έχουν παραμέτρους (τα δεδομένα που χρειάζονται ως input)*
  - *ένα block εντολών μέσα σε άγκιστρα*

```
14
15 int max(int a, int b) {
16 if (a > b)
17     return a;
18 else
19     return b;
20 }
```

- Υπάρχουν 3 διαφορετικές ενέργειες που πρέπει να γίνουν για κάθε συνάρτηση:
  - ***δήλωση*** *(declaration)*
  - ***ορισμός*** *(definition)*
  - ***κλήση / χρήση*** *(call / use)*
- Πριν χρησιμοποιήσουμε μια συνάρτηση, θα πρέπει να την έχουμε δηλώσει και ορίσει προηγουμένως.

Συνάρτηση - 1η ενέργεια: δήλωση

- Πριν χρησιμοποιήσουμε μια συνάρτηση, θα πρέπει να την έχουμε δηλώσει και ορίσει προηγουμένως.
- Ας δούμε τι σημαίνει η κάθε ενέργεια:
  1. *δήλωση - declaration (γραμμή 4 του παραδείγματος):*
     - προσδιορίζει τις απαιτήσεις και το αποτέλεσμα, δηλαδή το "**ΤΙ**" εργασία επιτελεί η συνάρτηση
     - αποτελεί το "πρωτότυπο" της συνάρτησης (function prototype)
     - στο πρωτότυπο *(γραμμή 4 παραδείγματος)* δηλώνονται:
       - το **όνομα** της συνάρτησης (max στο παράδειγμα)
       - οι **τύποι των δεδομένων** που απαιτούνται ως **παράμετροι** (δύο int στο παράδειγμα)
       - ο **τύπος των δεδομένων** που **επιστρέφονται** από τη συνάρτηση (int στο παράδειγμα)
     - παρατηρήστε ότι στο τέλος της δήλωσης τερματίζουμε την εντολή κανονικά με ;

```c
/* simplefunction.c */
#include <stdio.h>

int max(int, int); /* function declaration */

int main() {
    int x, y, z;

    x = 4;
    y = 5;
    z = max(x, y);   /* function call */
    printf("max of %d and %d is %d\n", x, y, z);
}

int max(int a, int b) { /* function definition */
    if (a > b)
        return a;
    else
        return b;
}
```

Συνάρτηση - 2η ενέργεια: ορισμός

- Πριν χρησιμοποιήσουμε μια συνάρτηση, θα πρέπει να την έχουμε δηλώσει και ορίσει προηγουμένως.
- Ας δούμε τι σημαίνει η κάθε ενέργεια:
  - *2. ορισμός - definition (γραμμες 15-20 του παραδείγματος):*
    - προσδιορίζει τις εσωτερικές διεργασίες, δηλαδή το "**ΠΩΣ**" εκτελείται η εργασία
    - στην "επικεφαλίδα" της συνάρτησης *(γραμμή 15 παραδείγματος)* ορίζονται και ονοματίζονται οι παράμετροι (οι εσωτερικές μεταβλητές a και b του παραδείγματος)
    - το "σώμα" της συνάρτησης *(γραμμές 16-19 παραδείγματος)* περιλαμβάνει μέσα σε άγκιστρα {} **το block των εντολών που θα εκτελεστούν** κάθε φορά που καλούμε τη συνάρτηση
    - παρατηρούμε ότι οι παράμετροι (a, b στο παράδειγμα) χρησιμοποιούνται στον κώδικα ως ορισμένες μεταβλητές

```c
/* simplefunction.c */
#include <stdio.h>

int max(int, int); /* function declaration */

int main() {
    int x, y, z;

    x = 4;
    y = 5;
    z = max(x, y);   /* function call */
    printf("max of %d and %d is %d\n", x, y, z);
}

int max(int a, int b) { /* function definition */
    if (a > b)
        return a;
    else
        return b;
}
```

Συνάρτηση - 3η ενέργεια: κλήση (χρήση)

- Πριν χρησιμοποιήσουμε μια συνάρτηση, θα πρέπει να την έχουμε δηλώσει και ορίσει προηγουμένως.
- Ας δούμε τι σημαίνει η κάθε ενέργεια:
  - *3. κλήση / χρήση - call / use (γραμμή 11 του παραδείγματος):*
    - κάθε φορά που θέλουμε να χρησιμοποιήσουμε τη συνάρτηση, **την "καλούμε" με το όνομά της και τα ανάλογα ορίσματα (τιμές)**
    - τότε, οι εσωτερικές μεταβλητές της συνάρτησης παίρνουν τις τιμές των ορισμάτων της κλήσης
    - στο παράδειγμα, κατά την κλήση της συνάρτησης τα a και b παίρνουν τις τιμές 4 και 5 των x και y αντίστοιχα

```
 1  /* simplefunction.c */
 2  #include <stdio.h>
 3
 4  int max(int, int); /* function declaration */
 5
 6  int main() {
 7      int x, y, z;
 8
 9      x = 4;
10      y = 5;
11      z = max(x, y);   /* function call */
12      printf("max of %d and %d is %d\n", x, y, z);
13  }
14
15  int max(int a, int b) { /* function definition */
16      if (a > b)
17          return a;
18      else
19          return b;
20  }
```

Κατηγορίες συναρτήσεων

Οι συναρτήσεις μπορούν να χωριστούν σε 4 κατηγορίες, ανάλογα με το αν έχουν παραμέτρους και αν επιστρέφουν κάποια τιμή.

1. συναρτήσεις **χωρίς** παραμέτρους που **δεν επιστρέφουν** κάποια τιμή
   - *απλές συναρτήσεις*
   - *συχνά χρησιμοποιούμε τέτοιες συναρτήσεις για να εμφανίσουμε κάτι στην οθόνη (πχ. ένα μενού επιλογών)*
   - *χρησιμοποιούμε τη λέξη '**void**' για να δηλώσουμε το 'κενό' στις παραμέτρους και στον τύπο της επιστροφής*
   - *παρατηρήστε ότι δεν υπάρχει η λέξη '**return**': μετά την εκτέλεση της τελευταίας εντολής, η ροή του προγράμματος επιστρέφει.*

```c
/* simplemenu.c */
#include <stdio.h>

void simplemenu (void); /* void */

int main (){
    simplemenu();
}

void simplemenu (void) {
    printf("**** Menu ****\n");
    printf("* 1. Add     *\n");
    printf("* 2. Delete  *\n");
    printf("**************\n");
}
```

2. συναρτήσεις **με** παραμέτρους που **δεν επιστρέφουν** κάποια τιμή
   - *συχνά χρησιμοποιούμε τέτοιες συναρτήσεις για να υπολογίσουμε κάτι και να το εμφανίσουμε στην οθόνη*
   - *χρησιμοποιούμε τη λέξη '**void**' για να δηλώσουμε το 'κενό' στον τύπο της επιστροφής (η συνάρτηση δεν επιστρέφει κάτι)*
   - *για κάθε παράμετρο, δηλώνουμε τον τύπο και το όνομα (που θα χρησιμοποιηθεί εσωτερικά)*
   - *κατά την κλήση, δίνουμε το σωστό πλήθος ορισμάτων, με τη σωστή σειρά*
   - *το **όρισμα** είναι η τιμή που θα δοθεί για κάθε **παράμετρο***
   - *στο παράδειγμα δεξιά: 4 και 9 είναι τα ορίσματα για τις παραμέτρους a και b αντίστοιχα*

```c
/* function - no return - with parameters */
#include <stdio.h>

void print_avg (int, int); /* prototype */

int main (){
    ...
    print_avg(4, 9);
    ...
}

void print_avg (int a, int b) {
    float average;

    average = (a + b) / 2.0;
    printf("average of %d and %d: %.2f\n", a, b, average);
}
```

Κατηγορίες συναρτήσεων

Οι συναρτήσεις μπορούν να χωριστούν σε 4 κατηγορίες, ανάλογα με το αν έχουν παραμέτρους και αν επιστρέφουν κάποια τιμή.

3. συναρτήσεις **χωρίς** παραμέτρους που **επιστρέφουν** κάποια τιμή
    - *χρησιμοποιούνται σπάνια*
    - *μια περίπτωση χρήσης τους είναι η εξειδίκευση μιας έτοιμης συνάρτησης βιβλιοθήκης*
    - *παράδειγμα: η χρήση της συνάρτησης βιβλιοθήκης παραγωγής τυχαίων αριθμών `rand()` για την προσομοίωση ζαριού (τυχαίοι αριθμοί από το 1 έως το 6)*
    - *αυτή η μέθοδος περιβάλλει με κώδικα την ήδη υπάρχουσα συνάρτηση για να την εξειδικεύσει*
    - *σημείωση: τις συναρτήσεις `time()`, `rand()` και `srand()` θα τις δούμε ξεχωριστά*

```c
/* rand() wrapper function *
 *  wraps around rand()    *
 *  to give a roll of dice */
#include <stdio.h>
#include <time.h>   /* for time() */
#include <stdlib.h> /* for srand() & rand() */

int rollDice (void);

int main () {
    printf("roll: %d\n", rollDice() );
}

int rollDice (void) {
    srand(time(NULL));
    return rand() % 6 + 1;
}
```

4. συναρτήσεις **με** παραμέτρους που **επιστρέφουν** κάποια τιμή
    - *η πιο συνηθισμένη κατηγορία συναρτήσεων:*
      δέχονται κάποιες τιμές, κάνουν υπολογισμούς με αυτές και επιστρέφουν ένα αποτέλεσμα
    - *πολλές φορές καταχωρούμε την τιμή που επιστρέφεται σε μια μεταβλητή ίδιου τύπου με τον τύπο της επιστροφής (όπως στη γραμμή 11 του παραδείγματος)*
    - *στο παράδειγμα δεξιά, τα ορίσματα x και y (γραμμή 11) αντιστοιχούν στις παραμέτρους a και b της συνάρτησης (γραμμή 15)*

```c
/* simplefunction.c */
#include <stdio.h>

int max (int, int); /* function declaration */

int main (){
    int x, y, z;

    x = 4;
    y = 5;
    z = max(x,y);   /* function call */
    printf ("max of %d and %d is %d\n", x, y, z);
}

int max (int a, int b) { /* function definition */
    if (a > b)
        return a;
    else
        return b;
}
```

Ένα πλήρες πρόγραμμα με συναρτήσεις

- Πρακτική εφαρμογή κώδικα με συναρτήσεις
  - *για την καλύτερη οργάνωση ενός προγράμματος, συνηθίζουμε να "σπάμε" τις διάφορες διεργασίες ενός προγράμματος σε συναρτήσεις*
  - *αυτός ο καταμερισμός βοηθά στην απλοποίηση του κώδικα και στην ευκολότερη διαχείριση*
- Προγραμματισμός σε φάσεις I
  - *πολλές φορές γράφουμε ένα πρόγραμμα σε φάσεις*
  - *ξεχωρίζουμε τις διεργασίες και αναθέτουμε κάθε διεργασία σε μια συνάρτηση*
  - *στη συνέχεια γράφουμε τον κώδικα, δημιουργώντας κενές συναρτήσεις προσομοίωσης (stub functions: διαβάστε το σχετικό άρθρο)*
  - *σε κάθε κενή συνάρτηση συνήθως γράφουμε ως σχόλιο τη λειτουργία της και βάζουμε ένα μήνυμα να γράφεται στην οθόνη (για δοκιμή)*
- Προγραμματισμός σε φάσεις II
  - *για κάθε συνάρτηση αντικαθιστούμε τη λειτουργία προσομοίωσης με την πραγματική λειτουργία και δοκιμάζουμε*
  - *στο τέλος θα έχουμε το πλήρες πρόγραμμα*

Ένα πλήρες πρόγραμμα με συναρτήσεις

Δείτε ένα πρόγραμμα με τις συναρτήσεις προσομοίωσης:
(Αν δεν φαίνεται ολόκληρο, κάντε scroll δεξιά)

```c
/* stubfunction.c */
/* Programma poy ypologizei to embado orismenvn
 * geometrikvn sxhmatvn */
#include <stdio.h>

/* function declarations */
void showMenu(void);
int getChoice(void);
void rectangularArea(void);
void diskArea(void);

int main() {
    /* basikh roh programmatos:
     * 1. typvnetai to menu
     * 2. zhtame kai pairnoume thn epilogh toy xrhsth
     * 3. analogvs ths epiloghs:
     *    kaloyme thn katallhlh synarthsh
     * */
    int choice;

    showMenu();
    choice = getChoice();
    switch (choice) {
        case 1: rectangularArea(); break;
        case 2: diskArea(); break;
        default: printf("Error\n"); break;
    }

    return 0;
}

/* function definitions */
void showMenu(void) {
    printf("showMenu was called. ");
    printf("It will print the menu.\n");
}

int getChoice(void) {
    printf("getAnswer was called. ");
    printf("It will return the user's choice.\n");
    return 0;
}

void rectangularArea (void) {
    /* ask for dimensions, calculate and print*/
    printf("rectangularArea was called.");
    printf("It will calculate and print the area ");
    printf("of a rectangular.\n");
}

```

Δείτε το πλήρες πρόγραμμα με τις συναρτήσεις του:
(Αν δεν φαίνεται ολόκληρο, κάντε scroll δεξιά)

```c
/* stubfunction.c */
/* Programma poy ypologizei to embado orismenvn
 * geometrikvn sxhmatvn */
#include <stdio.h>

/* function declarations */
void showMenu(void);
int getChoice(void);
void rectangularArea(void);
void diskArea(void);

int main() {
    /* basikh roh programmatos:
     * 1. typvnetai to menu
     * 2. zhtame kai pairnoume thn epilogh toy xrhsth
     * 3. analogvs ths epiloghs:
     *    kaloyme thn katallhlh synarthsh
     * */
    int choice;

    showMenu();
    choice = getChoice();
    switch (choice) {
        case 1: rectangularArea(); break;
        case 2: diskArea(); break;
        default: printf("Error\n"); break;
    }
    printf("End\n");

    return 0;
}

/* function definitions */
void showMenu(void) {
    printf("**************************\n");
    printf("* 1. Area of rectangular *\n");
    printf("* 2. Area of disk        *\n");
    printf("**************************\n");
}

int getChoice(void) {
    int ch;

    printf("Please give me your choice: ");
    scanf("%d", &ch);
    return ch;
}

void rectangularArea (void) {
    /* ask for dimensions, calculate and print*/
```

```c
void diskArea(void) {
    /* ask for radius, calculate and print*/
    printf("diskArea was called.");
    printf("It will calculate and print the area ");
    printf("of a circular disk.\n");
}
```

```c
    float length, width, area;

    printf("Please give me the dimensions of the rectangular: ");
    scanf("%f %f", &length, &width);
    area = length * width;
    printf("The area of the rectangular is %.3f\n", area);
}

void diskArea(void) {
    /* ask for radius, calculate and print*/
    const float PI = 3.14159;
    float radius, area;

    printf("Please give me the radius of the disk: ");
    scanf("%f", &radius);
    area = PI * radius * radius;
    printf("The area of the disk is %.3f\n", area);
}
```

Βιβλιοθήκες συναρτήσεων

- Με την εντολή #include <*stdio.h*> δίνουμε εντολή στον preprocessor να συμπεριλάβει στο πρόγραμμά μας τον κώδικα της βιβλιοθήκης `stdio`.
  - *Γενικά, στον Preprocessor δίνουμε οδηγίες (directives) με το σύμβολο #.*
  - *Λίστα οδηγιών (directives): #include #define #undef #ifdef #ifndef #if #else #elif #endif #error #pragma*
- Οι επικεφαλίδες (header files) των βιβλιοθηκών έχουν την κατάληξη `.h` και περιλαμβάνουν τα prototypes, δηλαδή τις δηλώσεις (declarations) των συναρτήσεων.
- Δείτε ένα τμήμα από το περιεχόμενο της κεφαλίδας `stdio.h`:
  - *παρατηρήστε τη δήλωση της printf() στη γραμμή 247:*

```
    ...
 38 #ifndef _STDIO_H_
 39 #define _STDIO_H_
    ...
246 void     perror(const char *);
247 int      printf(const char *, ...);
248 int      putc(int, FILE *);
249 int      putchar(int);
250 int      puts(const char *);
251 int      remove(const char *);
252 int      rename(const char *, const char *);
    ...
```

- Οι βιβλιοθήκες έτοιμων προγραμμάτων περιλαμβάνουν συναρτήσεις που:
  - *λειτουργούν σωστά*
    γιατί αυτές οι συναρτήσεις έχουν δοκιμαστεί εξονυχιστικά και είναι αρκετά γενικές και απλές στη χρήση.
  - *είναι τελειοποιημένες για αποδοτική λειτουργία*
    γιατί η κοινότητα των προγραμματιστών διαρκώς ελέγχει και βελτιώνει τον κώδικα
  - *είναι ήδη έτοιμες*
    για τις περισσότερες από τις συχνά χρησιμοποιούμενες λειτουργίες, εξοικονομώντας χρόνο και προσπάθεια
  - "*τρέχουν*" *παντού*
    γιατί είναι έτσι φτιαγμένες ώστε να λειτουργούν το ίδιο σε κάθε υπολογιστή