C: Από τη Θεωρία στην Εφαρμογή

Κεφάλαιο 11ο
Συναρτήσεις

Γ. Σ. Τσελίκης – Ν. 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**

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 επιστρεφόμενης τιμής πρέπει να ταιριάζει με τον τύπο που ορίστηκε να επιστρέφει η συνάρτηση στη δήλωσή της. Αν δεν ταιριάζουν, ο μεταγλωττιστής θα μετατρέψει την επιστρεφόμενη τιμή, αν αυτό είναι δυνατό, στον τύπο επιστροφής της συνάρτησης

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 (II)

```
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

```c
#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

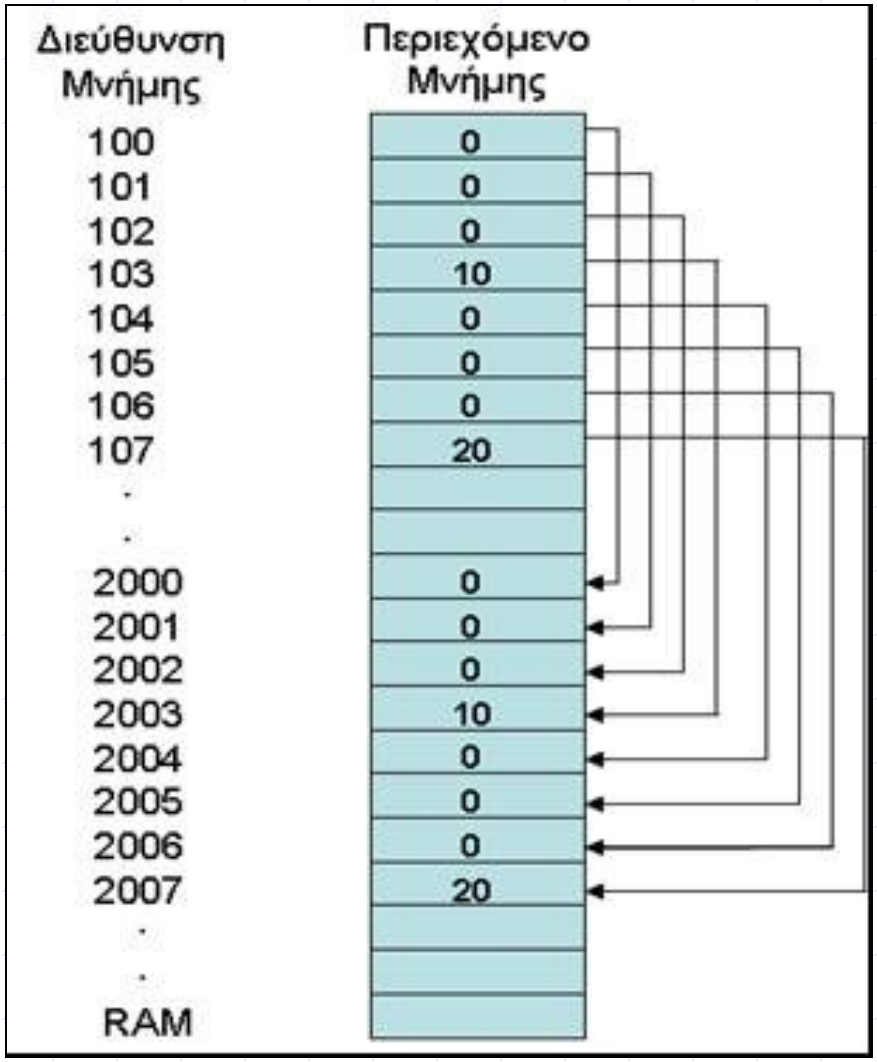
```c
#include <stdio.h>

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

```
#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

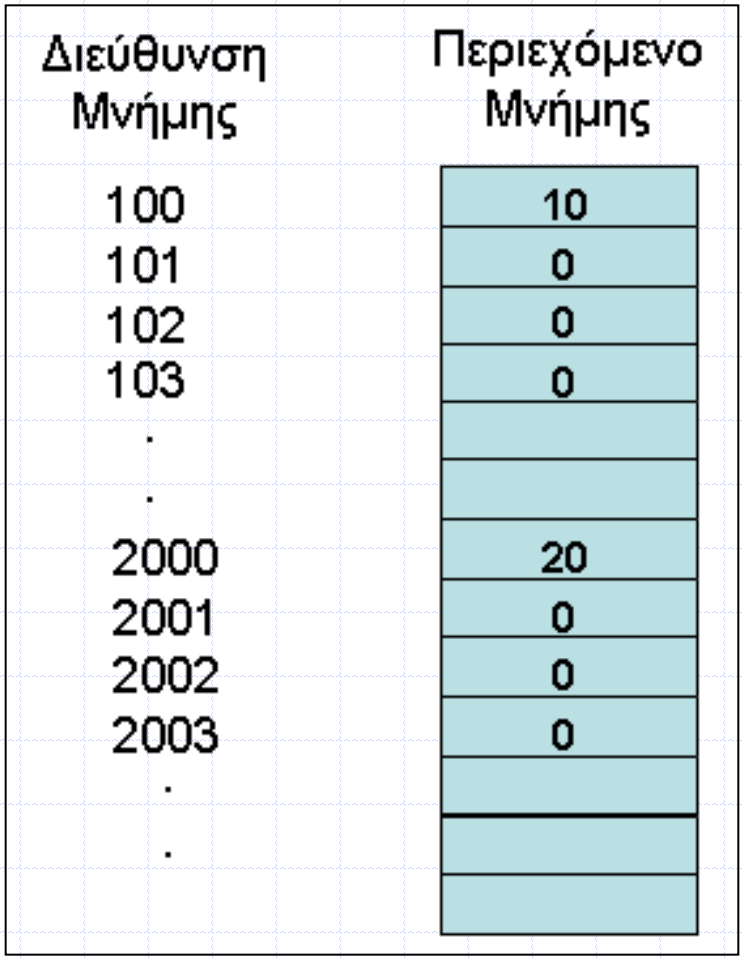
Παρατηρήσεις

- 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

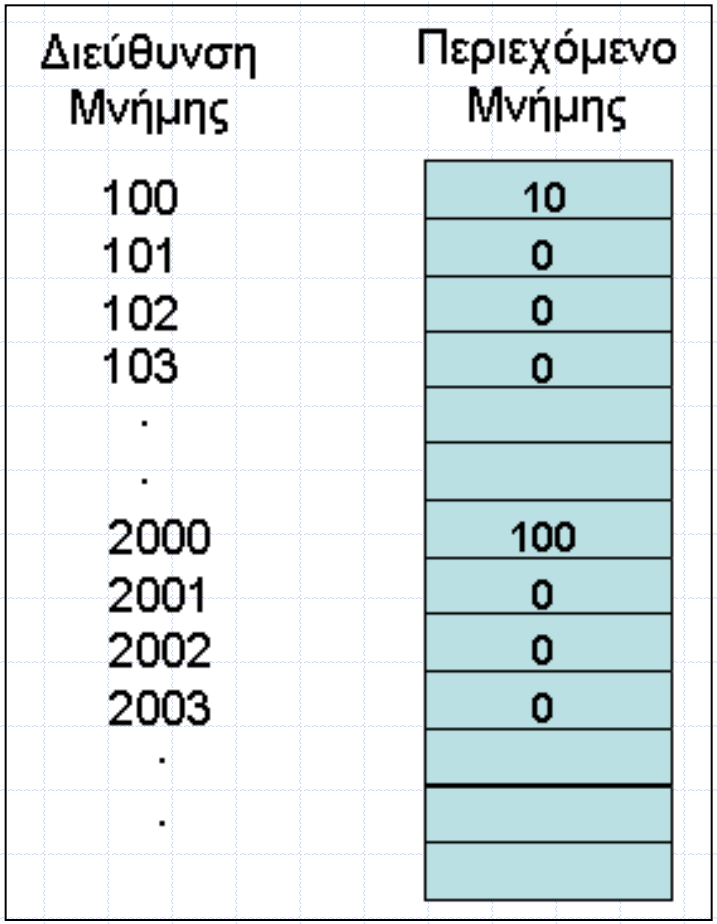
```c
#include <stdio.h>

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;
}
```

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
.	
.	

Έξοδος: 300 200

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 (Ι)

- 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`

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**