Διάλεξη 6η: Συναρτήσεις

Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Εισαγωγή στην Επιστήμη Υπολογιστών

Βασίζεται σε διαφάνειες του Κ. Παναγιωτάκη

Αλγόριθμοι και Προγράμματα

- Τα υπολογιστικά προβλήματα λύνονται εκτελώντας μια ακολουθία από εντολές με συγκεκριμένη σειρά
- Αλγόριθμος: ποιές είναι αυτές οι εντολές και η σειρά εκτέλεσής τους
- Πολλές φορές ο αλγόριθμος που λύνει ένα πρόβλημα χρησιμοποιεί άλλους αλγόριθμους για να λύσει μικρότερα επιμέρους προβλήματα:
  - Ο αλγόριθμος της διαίρεσης κάνει πολλές αφαιρέσεις
  - Ο αλγόριθμος της ύψωσης σε δύναμη κάνει πολλούς πολλαπλασιασμούς
- Μπορούμε να "γενικεύσουμε" τέτοιους υπο-αλγόριθμους και να τους "καλούμε" από όπου τους χρειαζόμαστε
- Συνάρτηση: ένα κομμάτι του προγράμματος που μπορούμε να "καλέσουμε" από άλλα σημεία του προγράμματος

Συναρτήσεις – Functions

- Σύνταξη:

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ... )
{
  declarations
  statements
}
```

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (`void`)

Συναρτήσεις – Functions

- Σύνταξη:

Τύπος του αποτελέσματος
που επιστρέφει η συνάρτηση

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ... )
{
  declarations
  statements
}
```

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστεφει αποτέλεσμα (`void`)

Συναρτήσεις – Functions

- Σύνταξη:

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ... )
{
  declarations
  statements
}
```

Όνομα της συνάρτησης: οποιοδήποτε δεκτό όνομα

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (void)

Συναρτήσεις – Functions

- Σύνταξη:

Μέσα σε παρενθέσεις τα ορίσματα της συνάρτησης

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ... )
{
  declarations
  statements
}
```

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (void)

Συναρτήσεις – Functions

- Σύνταξη:

Γενική μορφή

Τύπος του πρώτου ορίσματος

```
type name(type1 argument1, type2 argument2, ... )
{
  declarations
  statements
}
```

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (`void`)

Συναρτήσεις – Functions

- Σύνταξη:

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ... )
{
  declarations
  statements
}
```

Όνομα του πρώτου ορίσματος: λειτουργεί όπως μια τοπική μεταβλητή

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (void)

Συναρτήσεις – Functions

- Σύνταξη:

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ... )
{
  declarations
  statements
}
```

Μια συνάρτηση μπορεί να έχει πολλά ορίσματα (ή κανένα)

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (`void`)

Συναρτήσεις – Functions

- Σύνταξη:

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ... )
{
  declarations
  statements
}
```

Το κυρίως *σώμα* της συνάρτησης ξεκινάει με {

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (void)

Συναρτήσεις – Functions

- Σύνταξη:

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ... )
{
  declarations
  statements
}
```

και τελειώνει με }

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (void)

Συναρτήσεις – Functions

- Σύνταξη:

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ...)
{
  declarations
  statements
}
```

Ό,τι είναι μέσα σε { } λέγεται και *code block*

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (void)

Συναρτήσεις – Functions

- Σύνταξη:

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ...)
{
  declarations
  statements
}
```

Στην αρχή της συνάρτησης υπάρχουν οι δηλώσεις των τοπικών μεταβλητών

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (void)

Συναρτήσεις – Functions

- Σύνταξη:

Γενική μορφή

```
type name(type1 argument1, type2 argument2, ...)
{
  declarations
  statements
}
```

Μετά τις δηλώσεις μεταβλητών μπαίνουν οι εντολές της συνάρτησης

- Χωρίζουν το πρόγραμμα σε κομμάτια
- Οργάνωση του προγράμματος με “διαίρει και βασίλευε”
- Κάθε κομμάτι (συνάρτηση) λύνει ένα επιμέρους, μικρότερο πρόβλημα
- Building blocks για το πρόγραμμα
- Κάθε συνάρτηση έχει *ορίσματα* (είσοδο) και αποτέλεσμα (έξοδο)
  - Διαδικασία (από το Λύκειο): συνάρτηση που δεν επιστέφει αποτέλεσμα (void)

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```c
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Παράδειγμα: Μέτρηση ψηφίων

Χρησιμοποιείτε σχόλια για να περιγράψετε τη συνάρτηση

count-digits.c

```
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}

/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

Ένα σχόλιο αρχίζει με /*

```
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

και τελειώνει με */

```
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

Περίληψη: ποιό πρόβλημα λύνει η συνάρτηση;

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```c
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```c
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Τί παίρνει ως είσοδο;

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

Τί επιστρέφει στην έξοδο;

```
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

Η συνάρτηση επιστρέφει έναν ακέραιο αριθμό (integer)

```
#include <stdio.h>

/* Συνάρτηση digits: με δεδομένο ένα ψηφία
 * ενός αριθμού, επιστρέφει έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```c
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Η συνάρτηση ονομάζεται `count_digits`

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

Έχει ένα όρισμα x που είναι ακέραιος αριθμός

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

Το x είναι σαν μια τοπική μεταβλητή, που αρχικά περιέχει το όρισμα

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}

/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Δήλωση και αρχικοποίηση της τοπικής μεταβλητής count

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}

/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
    number, count_digits(number));
}
```

Αρχή των εντολών της συνάρτησης `count_digits`

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```c
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Η τελευταία εντολή επιστρέφει το αποτέλεσμα της συνάρτησης

Παράδειγμα: Μέτρηση ψηφίων

Η `main` είναι μια συνάρτηση χωρίς ορίσματα που επιστρέφει ακέραιο

count-digits.c

```
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Παράδειγμα: Μέτρηση ψηφίων

count_digits.c

Δήλωση της τοπικής μεταβλητής number

```
#include <stdio.h>

/* Συνάρτηση που επιστρέφει το πλήθος των ψηφίων
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}

/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

Αρχή των εντολών της συνάρτησης `main`

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού εκτός από το τελευταίο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}

/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```c
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Καλεί την `count_digits` με όρισμα `number`

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}

/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Ξεκινά η εκτέλεση της `count_digits`

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

Δημιουργείται το x σαν τοπική μεταβλητή και αρχικοποιείται με number

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```c
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
         number, count_digits(number));
}
```

Τελικά η `count_digits` επιστρέφει την τιμή που περιέχει η `count`

Παράδειγμα: Μέτρηση ψηφίων

count-digits.c

```c
#include <stdio.h>

/* Συνάρτηση digits: μετράει τα ψηφία
 * ενός αριθμού. Δέχεται έναν ακέραιο
 * αριθμό και επιστρέφει πόσα ψηφία έχει
 */
int count_digits(int x)
{
  int count = 0;

  while (x != 0) {
    x = x / 10;
    count++;
  }
  return count;
}
```

```c
/* Συνάρτηση main */
int main()
{
  int number;

  printf("Please give a number: ");
  scanf("%d", &number);
  printf("Number %d has %d digits.\n",
    number, count_digits(number));
}
```

Το αποτέλεσμα της `count_digits` το παίρνει η `printf` σαν όρισμα

Παράδειγμα: Δευτεροβάθμια εξίσωση

quadratic.c

```
/* Πρόγραμμα: quadratic.c
 * Λύνει μια δευτεροβάθμια εξίσωση
 */

#include <stdio.h>
#include <math.h>

/* function declarations */
float discriminant(float, float, float);
int solve_quadratic(float a, float b, float c);

/* function definitions */
int main()
{
  float a, b, c;

  printf("Please type 3 decimal numbers\n");
  scanf("%f %f %f",&a,&b,&c);
  printf("Quadratic equation: "
         "%fx^2 + (%f)x + (%f) = 0\n", a, b, c);
  if (a == 0) {
    printf("This is a first degree equation\n");
  } else {
    solve_quadratic(a, b, c);
  }
}

float discriminant(float a, float b, float c)
{
  return (b * b) -(4 * a * c);
}

int solve_quadratic(float a, float b, float c)
{
  float D, x1, x2;

  D = discriminant(a, b, c);

  if (D < 0) {
    printf("Discriminant is negative. "
           "There are no real solutions\n");
    return 0;
  } else if (D == 0) {
    x1 = -b / (2 * a);
    printf("One solution: %f\n", x1);
    return 1;
  } else {
    x1 = (-b + sqrt(D)) / (2 * a);
    x2 = (-b - sqrt(D)) / (2 * a);
    printf("Two solutions: %f, %f\n", x1, x2);
    return 2;
  }
}
```

Οργάνωση του προγράμματος

- Είναι γενικά ευκολότερο να διαχειριστούμε ένα πρόβλημα σπάζοντάς το σε υποπροβλήματα
- Κάθε υποπρόβλημα αντιστοιχεί σε μια συνάρτηση
- Πιο εύκολη διαχείριση κώδικα
  - Debugging: δοκιμή κάθε συνάρτησης ξεχωριστά, ευκολότερη εύρεση λαθών
  - Updating: ευκολότερη αλλαγή ενός αλγορίθμου με έναν αποδοτικότερο χωρίς να επηρρεάζεται το υπόλοιπο πρόγραμμα
- Επαναχρησιμοποίηση του κώδικα: χρήση υπαρχόντων συναρτήσεων σε νέα προγράμματα
- Αποφυγή επανάληψης του ίδιου κώδικα πολλές φορές
  - Το copy & paste αντιγράφει και τα λάθη...
- Αφαιρετική σκέψη (abstraction)
  - Κρύβει όσες λεπτομέρειες δεν χρειάζονται

Κλήση συναρτήσεων

- Η κλήση συνάρτησης είναι έκφραση

```
function_name(argument1, ..., argumentN)
```

- Η έκφραση αυτή έχει τύπο τον τύπο που επιστρέφει η συνάρτηση, και τιμή ό,τι επιστραφεί
  - `number = count_digits(10+10) + 2;`
- Για να εκτελεστεί η κλήση μιας συνάρτησης:
  1. Υπολογίζονται πρώτα οι τιμές των ορισμάτων
     - `number = count_digits(20) + 2;`
  2. Αντιγράφονται οι τιμές των ορισμάτων στις αντίστοιχες μεταβλητές της συνάρτησης
     - `x = 20;`
  3. Εκτελούνται οι εντολές της συνάρτησης μέχρι να εκτελεστεί κάποιο `return`
     - `return count;`
  4. Η κλήση συνάρτησης αντικαθίσταται με την τιμή που επιστρέφει
     - `number = 2 + 2;`

Διαδικασίες και συναρτήσεις

- Στο Λύκειο, είδατε συναρτήσεις που επιστρέφουν αποτέλεσμα, και διαδικασίες που δεν επιστρέφουν
- Η γλώσσα C έχει μόνο συναρτήσεις
- Μια συνάρτηση μπορεί να επιστρέφει αποτέλεσμα τύπου `void`, δηλαδή κενό

void.c

```
void f()
{
  printf("This is a test\n");
  return;
}
```

- Μια συνάρτηση που επιστρέφει `void` είναι σαν διεργασία, δεν μπορεί να καλεστεί σε έκφραση
  - Π.χ: δεν επιτρέπεται η εντολή `x = f() + 2`

Διαδικασίες και συναρτήσεις

- Στο Λύκειο, είδατε συναρτήσεις που επιστρέφουν αποτέλεσμα, και διαδικασίες που δεν επιστρέφουν
- Η γλώσσα C έχει μόνο συναρτήσεις
- Μια συνάρτηση μπορεί να επιστρέφει αποτέλεσμα τύπου `void`, δηλαδή κενό

void.c

```
void f()
{
  printf("This is a test\n");
  return;
}
```

Όταν η συνάρτηση επιστρέφει `void` τότε η εντολή `return` δεν παίρνει τιμή

- Μια συνάρτηση που επιστρέφει `void` είναι σαν διεργασία, δεν μπορεί να καλεστεί σε έκφραση
  - Π.χ: δεν επιτρέπεται η εντολή `x = f() + 2`

Συναρτήσεις βιβλιοθήκης

- Συναρτήσεις ορισμένες από τον προγραμματιστή
  - Όπως οι `f`, `count_digits`, `discriminant` παραπάνω
- Συναρτήσεις βιβλιοθήκης
  - Έτοιμες συναρτήσεις που μπορούμε να χρησιμοποιήσουμε
  - Όπως οι `printf` και `scanf`
- Βιβλιοθήκες έτοιμων συναρτήσεων
  - Η βιβλιοθήκη της γλώσσας C
  - Βιβλιοθήκες συναρτήσεων διαθέσιμες στο διαδίκτυο

Η βιβλιοθήκη της C

#include

```
int abs(int);
void exit(int);
int rand(void);
void srand(unsigned int);
```

- Μπορείτε να δείτε αναλυτικά όλες τις συναρτήσεις βιβλιοθήκης στο manual page: `$ man stdlib.h`
- Manual pages των επιμέρους συναρτήσεων: `$ man p rand`
- Manual page της εντολής man: `$ man man`
- Αντίστοιχα για Input/Output: `$ man stdio.h`

Η βιβλιοθήκη της C

Η απόλυτη τιμή ενός ακεραίου

#include

```
int abs(int);
void exit(int);
int rand(void);
void srand(unsigned int);
```

- Μπορείτε να δείτε αναλυτικά όλες τις συναρτήσεις βιβλιοθήκης στο manual page: `$ man stdlib.h`
- Manual pages των επιμέρους συναρτήσεων: `$ man p rand`
- Manual page της εντολής man: `$ man man`
- Αντίστοιχα για Input/Output: `$ man stdio.h`

Η βιβλιοθήκη της C

#include

```
int abs(int);
void exit(int);
int rand(void);
void srand(unsigned int);
```

Τερματίζει το πρόγραμμα επιστρέφοντας μια τιμή στο λειτουργικό σύστημα

- Μπορείτε να δείτε αναλυτικά όλες τις συναρτήσεις βιβλιοθήκης στο manual page: `$ man stdlib.h`
- Manual pages των επιμέρους συναρτήσεων: `$ man p rand`
- Manual page της εντολής man: `$ man man`
- Αντίστοιχα για Input/Output: `$ man stdio.h`

Η βιβλιοθήκη της C

#include

```
int abs(int);
void exit(int);
int rand(void);
void srand(unsigned int);
```

Δημιουργεί και επιστρέφει ένα ψευδοτυχαίο αριθμό από 0 μέχρι RAND_MAX με κανονική κατανομή

- Μπορείτε να δείτε αναλυτικά όλες τις συναρτήσεις βιβλιοθήκης στο manual page: `$ man stdlib.h`
- Manual pages των επιμέρους συναρτήσεων: `$ man p rand`
- Manual page της εντολής man: `$ man man`
- Αντίστοιχα για Input/Output: `$ man stdio.h`

Η βιβλιοθήκη της C

#include

```
int abs(int);
void exit(int);
int rand(void);
void srand(unsigned int);
```

Αρχικοποιεί τη γεννήτρια ψευδοτυχαίων αριθμών, αρκεί να την καλέσουμε μία φορά

- Μπορείτε να δείτε αναλυτικά όλες τις συναρτήσεις βιβλιοθήκης στο manual page: `$ man stdlib.h`
- Manual pages των επιμέρους συναρτήσεων: `$ man p rand`
- Manual page της εντολής man: `$ man man`
- Αντίστοιχα για Input/Output: `$ man stdio.h`

Η μαθηματική βιβλιοθήκη της C

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Η μαθηματική βιβλιοθήκη της C

"Ταβάνι", ο αμέσως μεγαλύτερος ακέραιος

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Η μαθηματική βιβλιοθήκη της C

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

Συνημίτονο

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Η μαθηματική βιβλιοθήκη της C

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

Εκθετικό e^x

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Η μαθηματική βιβλιοθήκη της C

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

Απολυτη τιμή

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Η μαθηματική βιβλιοθήκη της C

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

"Πάτωμα", ο αμέσως μικρότερος ακέραιος

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Η μαθηματική βιβλιοθήκη της C

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

Φυσικός λογάριθμος

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Η μαθηματική βιβλιοθήκη της C

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

Ύψωση σε δύναμη

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Η μαθηματική βιβλιοθήκη της C

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

Ημίτονο

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Η μαθηματική βιβλιοθήκη της C

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

Τετραγωνική ρίζα

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Η μαθηματική βιβλιοθήκη της C

#include

```
double ceil(double);
double cos(double);
double exp(double);
double fabs(double);
double floor(double);
double log(double);
double pow(double, double);
double sin(double);
double sqrt(double);
double tan(double);
```

Εφαπτομένη

- Το manual page της μαθηματικής βιβλιοθήκης:
  ```
  $ man math.h
  $ man pow
  ```
- Στο compile και link, προσθέτουμε `-lm` όταν καλούμε το gcc:
  ```
  $ gcc power.c -o power.exe -lm
  ```

Ορισμός συναρτήσεων

- Οι δηλώσεις μεταβλητών μέσα σε code blocks δημιουργούν *τοπικές* μεταβλητές
  - Αυτές οι μεταβλητές χάνονται στο τέλος του block
  - Εκτός αν είναι `static` μεταβλητές
- *Δεν* μπορούν να οριστούν συναρτήσεις μέσα σε συναρτήσεις

Λάθος πρόγραμμα!

```
int main()
{
  void f() {
    printf("This is not valid C.\n");
  }
}
```

- Οποιαδήποτε συνάρτηση μπορεί να καλέσει όποια άλλη συνάρτηση έχει δηλωθεί ή οριστεί
  - Ακόμη και τον εαυτό της (αναδρομή)
  - Αν δεν έχουμε ορίσει ή δηλώσει μια συνάρτηση δεν μπορούμε να την καλέσουμε

Δηλώσεις συναρτήσεων

- Για να χρησιμοποιήσουμε μια συνάρτηση πρέπει να την ορίσουμε ή να τη *δηλώσουμε*
- Δήλωση συνάρτησης (function prototype ή declaration)

Μορφή δήλωσης συνάρτησης

```
type name(type1, ..., typeN);
```

- Η δήλωση (declaration) μιας συνάρτησης διαφέρει από τον ορισμό (definition) της
  - Δεν χρειάζεται ονόματα παραμέτρων
  - Δεν υπάρχει “σώμα” της συνάρτησης (δηλώσεις τοπικών μεταβλητών και εντολές)
- Μετά τη δήλωση μπορούμε να την καλέσουμε

Δηλώσεις συναρτήσεων

- Για να χρησιμοποιήσουμε μια συνάρτηση πρέπει να την ορίσουμε ή να τη *δηλώσουμε*
- Δήλωση συνάρτησης (function prototype ή declaration)

Μορφή δήλωσης συνάρτησης

```
type name(type1, ..., typeN);
```

Ο τύπος που επιστρέφει η συνάρτηση (όπως στον ορισμό)

- Η δήλωση (declaration) μιας συνάρτησης διαφέρει από τον ορισμό (definition) της
  - Δεν χρειάζεται ονόματα παραμέτρων
  - Δεν υπάρχει “σώμα” της συνάρτησης (δηλώσεις τοπικών μεταβλητών και εντολές)
- Μετά τη δήλωση μπορούμε να την καλέσουμε

Δηλώσεις συναρτήσεων

- Για να χρησιμοποιήσουμε μια συνάρτηση πρέπει να την ορίσουμε ή να τη *δηλώσουμε*
- Δήλωση συνάρτησης (function prototype ή declaration)

Μορφή δήλωσης συνάρτησης

```
type name(type1, ..., typeN);
```

Το όνομα της συνάρτησης (όπως στον ορισμό)

- Η δήλωση (declaration) μιας συνάρτησης διαφέρει από τον ορισμό (definition) της
  - Δεν χρειάζεται ονόματα παραμέτρων
  - Δεν υπάρχει “σώμα” της συνάρτησης (δηλώσεις τοπικών μεταβλητών και εντολές)
- Μετά τη δήλωση μπορούμε να την καλέσουμε

Δηλώσεις συναρτήσεων

- Για να χρησιμοποιήσουμε μια συνάρτηση πρέπει να την ορίσουμε ή να τη *δηλώσουμε*
- Δήλωση συνάρτησης (function prototype ή declaration)

Μορφή δήλωσης συνάρτησης

```
type name(type1, ..., typeN);
```

Οι τύποι των ορισμάτων της συνάρτησης

- Η δήλωση (declaration) μιας συνάρτησης διαφέρει από τον ορισμό (definition) της
  - Δεν χρειάζεται ονόματα παραμέτρων
  - Δεν υπάρχει “σώμα” της συνάρτησης (δηλώσεις τοπικών μεταβλητών και εντολές)
- Μετά τη δήλωση μπορούμε να την καλέσουμε

Δηλώσεις συναρτήσεων

- Για να χρησιμοποιήσουμε μια συνάρτηση πρέπει να την ορίσουμε ή να τη *δηλώσουμε*
- Δήλωση συνάρτησης (function prototype ή declaration)

Μορφή δήλωσης συνάρτησης

```
type name(type1, ..., typeN);
```

Δεν χρειάζονται τα ονόματα των ορισμάτων

- Η δήλωση (declaration) μιας συνάρτησης διαφέρει από τον ορισμό (definition) της
  - Δεν χρειάζεται ονόματα παραμέτρων
  - Δεν υπάρχει “σώμα” της συνάρτησης (δηλώσεις τοπικών μεταβλητών και εντολές)
- Μετά τη δήλωση μπορούμε να την καλέσουμε

Δηλώσεις συναρτήσεων

- Για να χρησιμοποιήσουμε μια συνάρτηση πρέπει να την ορίσουμε ή να τη *δηλώσουμε*
- Δήλωση συνάρτησης (function prototype ή declaration)

Μορφή δήλωσης συνάρτησης

```
type name(type1, ..., typeN);
```

Η δήλωση συνάρτησης τελειώνει με semicolon ;

- Η δήλωση (declaration) μιας συνάρτησης διαφέρει από τον ορισμό (definition) της
  - Δεν χρειάζεται ονόματα παραμέτρων
  - Δεν υπάρχει “σώμα” της συνάρτησης (δηλώσεις τοπικών μεταβλητών και εντολές)
- Μετά τη δήλωση μπορούμε να την καλέσουμε

Δηλώσεις συναρτήσεων

- Για να χρησιμοποιήσουμε μια συνάρτηση πρέπει να την ορίσουμε ή να τη *δηλώσουμε*
- Δήλωση συνάρτησης (function prototype ή declaration)

Μορφή δήλωσης συνάρτησης

```
type name(type1, ..., typeN);
```

- Η δήλωση (declaration) μιας συνάρτησης διαφέρει από τον ορισμό (definition) της
  - Δεν χρειάζεται ονόματα παραμέτρων
  - Δεν υπάρχει “σώμα” της συνάρτησης (δηλώσεις τοπικών μεταβλητών και εντολές)
- Μετά τη δήλωση μπορούμε να την καλέσουμε

Δηλώσεις συναρτήσεων (2)

- Τοποθετούμε τις δηλώσεις όλων των συναρτήσεων στην αρχή του προγράμματος
- Χρειάζεται υποχρεωτικά αν χρησιμοποιούμε συναρτήσεις πριν τις ορίσουμε
- Καλή πρακτική:
  - Ελέγχουμε τη σωστή χρήση των συναρτήσεων: σωστός τύπος ορισμάτων, επιστρεφόμενης τιμής
  - Μπορούμε να δούμε με μια ματιά τις συναρτήσεις του προγράμματος
  - Μπορούμε να καλέσουμε όλες τις συναρτήσεις από όλες τις άλλες άσχετα με τη σειρά που ορίζονται

Παράδειγμα: Ηλεκτρονικό ζάρι

die.c

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

/* declarations */
int throw_die(unsigned int);
void init();
int main();

/* definitions */

int main() {
  int result;

  init();
  printf("I'm throwing a 6-sided die.\n");
  result = throw_die(6);
  printf("%d came out.\n", result);
}
```

```c
/* Συνάρτηση init
 * Αρχικοποιεί τη γεννήτρια
 * τυχαίων αριθμών
 */
void init() {
  int seconds_since_1970 = time(0);
  srand(seconds_since_1970);
}

/* Συνάρτηση throw_die
 * Προσομοιώνει ένα ζάρι
 * Δέχεται τον αριθμό των πλευρών
 * του ζαριού
 * Επιστρέφει ένα τυχαίο αριθμό από
 * 1 μέχρι sides
 */
int throw_die(unsigned int sides) {
  return (rand() % sides) + 1;
}
```