Συναρτήσεις στη C++

ΠΕΡΙΕΧΟΜΕΝΑ

- Μαθηματικές Συναρτήσεις (Math Library)
- Συναρτήσεις
- Header Files
- Γεννήτρια τυχαίων αριθμών
- Χαρακτηριστικά Μεταβλητών
  - Storage Classes
  - Κανόνες Εμβέλειας (Scope Rules)
- Αναδρομή
- Συναρτήσεις με κενές λίστες παραμέτρων
- Inline Functions
- Αναφορές Παραμέτρων
- Αναφορές Μεταβλητών
- Default Ορίσματα
- Μοναδιαίος τελεστής Scope Resolution ::
- Υπερφόρτωση Συναρτήσεων (Function Overloading)
- Περίγραμμα Συναρτήσεων (Function Template)
- Ασκήσεις

Μαθηματικές Συναρτήσεις (Math Library)

- … για υλοποίηση απλών μαθηματικών λειτουργιών
  - Απαιτείται η συμπερίληψη του header file **<cmath>**
- Παράδειγμα

```
cout << sqrt( 900.0 );
```

  - Το αποτέλεσμα είναι η εκτύπωση της τιμής 30.0
  - Όλες οι συναρτήσεις της math library επιστρέφουν **double**

Μαθηματικές Συναρτήσεις (Math Library)

- Το όρισμα μπορεί να είναι:
  - Σταθερά
    - `sqrt( 4 );`
  - Μεταβλητή
    - `sqrt( x );`
  - Έκφραση
    - `sqrt( sqrt( x ) ) ;`
    - `sqrt( 3 - 6x );`

Method	Description	Example
`ceil( x )`	rounds *x* to the smallest integer not less than *x*	`ceil( 9.2 )` is `10.0` `ceil( -9.8 )` is `-9.0`
`cos( x )`	trigonometric cosine of *x* (*x* in radians)	`cos( 0.0 )` is `1.0`
`exp( x )`	exponential function *ex*	`exp( 1.0 )` is `2.71828` `exp( 2.0 )` is `7.38906`

`fabs( x )`	absolute value of *x*	`fabs( 5.1 )` is `5.1`
		`fabs( 0.0 )` is `0.0`
		`fabs( -8.76 )` is `8.76`

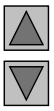
`floor( x )`	rounds *x* to the largest integer not greater than *x*	`floor( 9.2 )` is `9.0` `floor( -9.8 )` is `-10.0`
`fmod( x, y )`	remainder of *x*/*y* as a floating-point number	`fmod( 13.657, 2.333 )` is `1.992`
`log( x )`	natural logarithm of *x* (base *e*)	`log( 2.718282 )` is `1.0` `log( 7.389056 )` is `2.0`
`log10( x )`	logarithm of *x* (base 10)	`log10( 10.0 )` is `1.0` `log10( 100.0 )` is `2.0`
`pow( x, y )`	*x* raised to power *y* (*xy*)	`pow( 2, 7 )` is `128` `pow( 9, .5 )` is `3`
`sin( x )`	trigonometric sine of *x* (*x* in radians)	`sin( 0.0 )` is `0`
`sqrt( x )`	square root of *x*	`sqrt( 900.0 )` is `30.0` `sqrt( 9.0 )` is `3.0`
`tan( x )`	trigonometric tangent of *x* (*x* in radians)	`tan( 0.0 )` is `0`

Fig. 3.2 Math library functions.

3.5 Ορισμοί Συναρτήσεων

- Πρωτότυπο συνάρτησης
  - Ενημερώνει το μεταγλωττιστή για τον τύπο των παραμέτρων και τον επιστρεφόμενο τύπο της συνάρτησης
  - **`int square( int );`**
    - Συνάρτηση που λαμβάνει **`int`** και επιστρέφει **`int`**
    - Αποτελεί *τη δήλωση της συνάρτησης* (function declaration)

- Κλήση συνάρτησης
  - **`square(x);`**
  - Παρενθέσεις, τελεστής κλήσης συνάρτησης
    - Πέρασμα παραμέτρου x
    - Η συνάρτηση δέχεται το δικό της αντίγραφο των παραμέτρων
  - Αφού τερματίσει περνάει πίσω το αποτέλεσμα



Outline

fig03_03.cpp (1 of 2)

```
1  // Fig. 3.3: fig03_03.cpp
2  // Creating and using a programmer-defined function.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int square( int );   // functi
9
10 int main()
11 {
12    // loop 10 times and calculate and ou
13    // square of x each time
14    for ( int x = 1; x <= 10; x++ )
15       cout << square( x ) << "  ";  // f
16
17    cout << endl;
18
19    return 0;  // indicates successful termination
20
21 } // end main
22
```

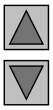
Πρωτότυπο: ορίζει τους τύπους δεδομένων και επιστρεφόμενες τιμές. Η **`square`** αναμένει **`int`**, επιστρέφει **`int`**.

Οι παρενθέσεις **`()`** προκαλούν κλήση συνάρτησης. Όταν τελειώσουν επιστρέφεται το αποτέλεσμα.

```
23 // square function definition returns square of an integer
24 int square( int y )  // y is a copy of argument to function
25 {
26    return y * y;      // returns square of y as an int
27
28 } // end function square
```

Ορισμός **square**. Το **y** είναι αντίγραφο του ορίσματος που περνάει. Επιστρέφει **y * y**.

```
1  4  9  16  25  36  49  64  81  100
```



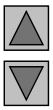
Outline

fig03_03.cpp (2 of 2)

fig03_03.cpp output (1 of 1)

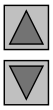

```
1  // Fig. 3.4: fig03_04.cpp
2  // Finding the maximum of three floating-point numbers.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  double maximum( double, double, double ); // function prototype
10
11 int main()
12 {
13    double number1;
14    double number2;
15    double number3;
16
17    cout << "Enter three floa
18    cin >> number1 >> number2 >> number3;
19
20    // number1, number2 and number3 are arguments to
21    // the maximum function call
22    cout << "Maximum is: "
23         << maximum( number1, number2, number3 ) << endl;
24
25    return 0;  // indicates successful termination
```

Η συνάρτηση **maximum** παίρνει 3 παραμέτρους (όλες **double**) και επιστρέφει **double**.



Outline

fig03_04.cpp (1 of 2)



Outline

```
26
27 } // end main
28
29 // function maximum definition;
30 // x, y and z are parameters
31 double maximum( double x, double y, double z )
32 {
33    double max = x;    // assume x is largest
34
35    if ( y > max )     // if y is larger,
36       max = y;        // assign y to max
37
38    if ( z > max )     // if z is larger,
39       max = z;        // assign z to max
40
41    return max;        // max is largest value
42
43 } // end function maximum
```

Λίστα παραμέτρων που χωρίζεται με κόμματα, για πολλές παραμέτρους.

fig03_04.cpp (2 of 2)

fig03_04.cpp output (1 of 1)

```
Enter three floating-point numbers: 99.32 37.3 27.1928
Maximum is: 99.32

Enter three floating-point numbers: 1.1 3.333 2.22
Maximum is: 3.333

Enter three floating-point numbers: 27.9 14.31 88.99
Maximum is: 88.99
```


3.6 Πρωτότυπα συναρτήσεων

- Αρχικές δηλώσεις των συναρτήσεων ώστε να μπορούν χρησιμοποιηθούν από πρόγραμμα
- Το πρωτότυπο πρέπει να ταιριάζει με τον ορισμό της συνάρτησης
  - Function prototype
    ```
    double maximum( double, double, double );
    ```
  - Definition
    ```
    double maximum( double x, double y, double z )
    {
      ...
    }
    ```
- Υπογραφή συνάρτησης
  - Το μέρος του πρωτοτύπου με όνομα και παραμέτρους
    - `double maximum( double, double, double );`

Υπογραφή συνάρτησης

Header Files (Αρχεία επικεφαλίδων)

- Περιέχουν
  - Πρωτότυπα συναρτήσεων
  - Ορισμούς τύπων και σταθερών
- Τα αρχεία επικεφαλίδων έχουν κατάληξη .h
  - Programmer-defined header files

```
#include "myheader.h"
```

- Library header files

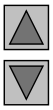
```
#include <cmath>
```


Γεννήτρια τυχαίων αριθμών

- **`rand`** function (**`<cstdlib>`**)
  - **`i = rand();`**
  - Παράγει έναν unsigned integer μεταξύ 0 και RAND_MAX (συνήθως 32767)
- Scaling and shifting
  - Παράδειγμα

```
i = rand() % 6 + 1;
```

    - "**`Rand() % 6`**" παράγει έναν αριθμό μεταξύ **`0`** και **`5`** (scaling)
    - "**`+ 1`**" δίνει το διάστημα 1 έως 6 (shift)

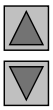


Outline

fig03_07.cpp (1 of 2)

```
1  // Fig. 3.7: fig03_07.cpp
2  // Shifted, scaled integers produced by 1 + rand() % 6.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 #include <cstdlib>   // contains function prototype for rand
13
14 int main()
15 {
16    // loop 20 times
17    for ( int counter = 1; counter <= 20; counter++ )
18
19       // pick random number from 1 to 6 and output it
20       cout << setw( 10 ) << ( 1 + rand() % 6 );
21
22       // if counter divisible by 5, begin new line of output
23       if ( counter % 5 == 0 )
24          cout << endl;
25
26    } // end for structure
```

Output of **`rand()`** scaled and shifted to be a number between 1 and 6.



Outline

fig03_07.cpp (2 of 2)

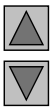
```
27
28    return 0;  // indicates successful termination
29
30 } // end main
```

fig03_07.cpp output (1 of 1)

```
         6         6         5         5         6
         5         1         1         5         3
         6         6         2         4         2
         6         2         3         4         1
```

Γεννήτρια τυχαίων αριθμών

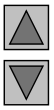
- Το επόμενο πρόγραμμα
  - Δείχνει την κατανομή των παραγόμενων αριθμών από τη **`rand()`**
  - Εξομοιώνει 6000 ρίψεις ενός ζαριού
  - Τυπώνει τα στατιστικά των ρίψεων (πόσες φορές φέραμε 1, 2, 3, κ.ο.κ.)
  - Αναμένουμε περίπου 1000 εμφανίσεις κάθε δυνατού αποτελέσματος



Outline

fig03_08.cpp
(1 of 3)

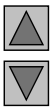
```
1  // Fig. 3.8: fig03_08.cpp
2  // Roll a six-sided die 6000 times.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 #include <cstdlib>   // contains function prototype for rand
13
14 int main()
15 {
16    int frequency1 = 0;
17    int frequency2 = 0;
18    int frequency3 = 0;
19    int frequency4 = 0;
20    int frequency5 = 0;
21    int frequency6 = 0;
22    int face;  // represents one roll of the die
23
```



Outline

fig03_08.cpp
(2 of 3)

```
24    // loop 6000 times and summarize results
25    for ( int roll = 1; roll <= 6000; roll++ ) {
26       face = 1 + rand() % 6;  // random number from 1 to 6
27
28       // determine face value and increment appropriate counter
29       switch ( face ) {
30
31          case 1:           // rolled 1
32             ++frequency1;
33             break;
34
35          case 2:           // rolled 2
36             ++frequency2;
37             break;
38
39          case 3:           // rolled 3
40             ++frequency3;
41             break;
42
43          case 4:           // rolled 4
44             ++frequency4;
45             break;
46
47          case 5:           // rolled 5
48             ++frequency5;
49             break;
```

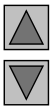
Outline

fig03_08.cpp
(3 of 3)

```
50
51          case 6:           // rolled 6
52             ++frequency6;
53             break;
54
55          default:          // invalid value
56             cout << "Program should never get here!";
57
58       } // end switch
59
60    } // end for
61
62    // display results in tabula
63    cout << "Face" << setw( 13 )
64         << "\n    1" << setw( 13 ) << frequency1
65         << "\n    2" << setw( 13 ) << frequency2
66         << "\n    3" << setw( 13 ) << frequency3
67         << "\n    4" << setw( 13 ) << frequency4
68         << "\n    5" << setw( 13 ) << frequency5
69         << "\n    6" << setw( 13 ) << frequency6 << endl;
70
71    return 0;  // indicates successful termination
72
73 } // end main
```

Η default περίπτωση περιλαμβάνεται σαν δείγμα καλού προγρ. Στυλ, αν και το πρόγραμμα δεν θα φτάσει εδώ.

```
Face    Frequency
   1         1003
   2         1017
   3          983
   4          994
   5         1004
   6          999
```



Outline

fig03_08.cpp
output (1 of 1)

Γεννήτρια τυχαίων αριθμών

- Η χρήση της rand() σε επαναλαμβανόμενες εκτελέσεις του προγράμματος
  - Αποδίδει την ίδια ακολουθία αριθμών
- Για να πάρουμε διαφορετικές ακολουθίες αριθμών
  - Χρησιμοποιούμε μια τιμή φύτρο (seed value)
    - Αντιστοιχεί σε τυχαίο σημείο εκκίνησης της ακολουθίας
    - Το ίδιο seed θα αποδώσει την ίδια ακολουθία
  - **`srand(seed);`**
    - **`<cstdlib>`**
    - Χρησιμοποιείται πριν από τη **`rand()`** για προσδιορίσει το τυχαίο σημείο εκκίνησης

Γεννήτρια τυχαίων αριθμών

- Μπορούμε να χρησιμοποιήσουμε την τρέχουσα ώρα ως φύτρο (seed)
  - `srand( time( 0 ) );`
  - `time( 0 );`
    - `<ctime>`
    - Επιστρέφει την τρέχουσα ώρα σε δευτερόλεπτα
- General shifting and scaling
  - *Number* **=** *shiftingValue* **+ rand()** **%** *scalingFactor*
  - shiftingValue = first number in desired range
  - scalingFactor = width of desired range

Χαρακτηριστικά Μεταβλητών

- Οι μεταβλητές χαρακτηρίζονται από διάφορες ιδιότητες
  - όνομα, τύπο δεδομένων, μέγεθος, τιμή
  - Τάξη αποθήκευσης (Storage Class)
    - Για ποιο διάστημα η μεταβλητή υπάρχει στη μνήμη (Duration)
  - Εμβέλεια (Scope)
    - Το τμήμα του προγράμματος που μια αναφορά στη μεταβλητή είναι έγκυρη
  - Συνδεσιμότητα (Linkage)
    - Όταν ένα πρόγραμμα υλοποιείται σε πολλά αρχεία, ποια αρχεία μπορούν να τη χρησιμοποιήσουν
    - Αν δηλαδή είναι ορατές στον linker ή όχι

Storage Classes

- **Automatic** μεταβλητές
  - Η μεταβλητή δημιουργείται όταν ο έλεγχος εισέρχεται σ' ένα block εντολών, π.χ. `for (int i = 0;…)`
  - Η μεταβλητή καταστρέφεται όταν ο έλεγχος εξέρχεται από ένα block εντολών
  - Μόνο οι τοπικές μεταβλητές συναρτήσεων μπορούν να είναι automatic
    - Είναι automatic εξ' ορισμού
    - Μπορεί να χρησιμοποιηθεί το keyword **`auto`**
- ~~**`register`** keyword~~ (καταργήθηκε στη C++ 11)
  - **Προτρέπει** τον μεταγλωττιστή να αποθηκεύσει τη μεταβλητή σ΄ έναν high-speed register
  - Καλή τακτική για δείκτες που προσπελάζονται συχνά (loop counters)
  - Ένας καλός μεταγλωττιστής μπορεί να προλάβει τον προγραμματιστή! (Βελτιστοποίηση)

Storage Classes

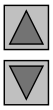
- **Static** μεταβλητές
  - Οι μεταβλητές εξακολουθούν να υπάρχουν καθ' όλη τη διάρκεια ζωής του προγράμματος
  - Για τις συναρτήσεις το όνομα υπάρχει καθ' όλη τη διάρκεια ζωής του προγράμματος
- **`static`** keyword
  - Εάν εφαρμοστεί σε τοπικές μεταβλητές συναρτήσεων
  - Διατηρεί την τιμή της μεταβλητής μεταξύ διαφορετικών κλήσεων της συνάρτησης
  - Η κατάσταση της μεταβλητής είναι γνωστή μόνο στη συνάρτηση
- **`extern`** keyword
  - **Default** για global variables/functions (αλλά όχι για const)
  - Διαχωρίζει τη δήλωση από τον ορισμό (π.χ. σε ξεχωριστό αρχείο)

Κανόνες Εμβέλειας (Scope Rules)

- Εμβέλεια (Scope)
  - Η περιοχή του προγράμματος που ένα όνομα (π.χ. μεταβλητής) μπορεί να χρησιμοποιηθεί
- Εμβέλεια αρχείου (File scope)
  - Ορίζεται εκτός μιας συνάρτησης, διαθεσιμότητα για όλες τις συναρτήσεις
  - Καθολικές μεταβλητές, ορισμοί και πρωτότυπα συναρτήσεων
  - **Static file scope:** δεν μπορεί να χρησιμοποιηθεί σε άλλα αρχεία (internal linkage)
  - **Non-static file scope:** μπορεί να χρησιμοποιηθεί σε άλλα αρχεία (external linkage, μέσω forward declaration - extern)
- Εμβέλεια συνάρτησης (Function scope)
  - Η περιοχή πρόσβασης περιορίζεται μόνο στη συνάρτηση που ορίζεται το όνομα

Κανόνες Εμβέλειας (Scope Rules)

- Εμβέλεια μπλοκ (Block scope)
  - Ξεκινά από το σημείο δήλωσης του ονόματος και τερματίζει στο κλείσιμο του μπλοκ που υποδηλώνεται με το σύμβολο `}`
  - Τοπικές μεταβλητές, παράμετροι συναρτήσεων
  - **`οι static`** μεταβλητές επίσης μπορεί να έχουν εμβέλεια μπλοκ
    - Άλλο το Storage class και άλλο η εμβέλεια
- Εμβέλεια πρωτότυπου συνάρτησης
  - Λίστα παραμέτρων
  - Τα ονόματα στο πρωτότυπο είναι προαιρετικά
    - Ο μεταγλωττιστής τα αγνοεί



Outline

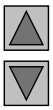
fig03_12.cpp
(1 of 5)

```
1  // Fig. 3.12: fig03_12.cpp
2  // A scoping example.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  void useLocal( void );
9  void useStaticLocal( void );
10 void useGlobal( void );        // function prototype
11
12 int x = 1;         // global variable
13
14 int main()
15 {
16    int x = 5;   // local variable to main
17
18    cout << "local x in main's outer scope is " << x << endl;
19
20    { // start new scope
21
22       int x = 7;
23
24       cout << "local x in main's inner scope is " << x << endl;
25
26    } // end new scope
```

Καθολική εμβέλεια

Τοπική εμβέλεια μέσα στη συνάρτηση

Εμβέλεια κατά το χρόνο ζωής του μπλόκ.



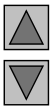
Outline

fig03_12.cpp
(2 of 5)

```
27
28    cout << "local x in main's outer scope is " << x << endl;
29
30    useLocal();       // useLocal has local x
31    useStaticLocal(); // useStaticLocal has static local x
32    useGlobal();      // useGlobal uses global x
33    useLocal();       // useLocal reinitializes its local x
34    useStaticLocal(); // static local x retains its prior value
35    useGlobal();      // global x also retains its value
36
37    cout << "\nlocal x in main is " << x << endl;
38
39    return 0;   // indicates successful termination
40
41 } // end main
42
```

```
43 // useLocal reinitializes local variable x during each call
44 void useLocal( void )
45 {
46    int x = 25;  // initialized each time useLocal is called
47
48    cout << endl << "local x is " << x
49         << " on entering useLocal" << endl;
50    ++x;
51    cout << "local x is " << x
52         << " on exiting useLocal" << endl;
53
54 } // end function useLocal
55
```

Αυτόματη μεταβλητή (τοπική μεταβλητή συνάρτησης). Καταστρέφεται με την έξοδο από τη συνάρτηση και επαναρχικοποιείται όταν αυτή ξεκινά.

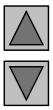


Outline

fig03_12.cpp (3 of 5)


```
56 // useStaticLocal initializes static local variable x only the
57 // first time the function is called; value of x is saved
58 // between calls to this function
59 void useStaticLocal( void )
60 {
61    // initialized only first time useStaticLocal is called
62    static int x = 50;
63
64    cout << endl << "local static x is " << x
65         << " on entering useStaticLocal" << endl;
66    ++x;
67    cout << "local static x is " << x
68         << " on exiting useStaticLocal" <<
69
70 } // end function useStaticLocal
71
```

Στατική τοπική μεταβλητή. Αρχικοποιείται μόνο μια φορά και διατηρεί την τιμή της μεταξύ των κλήσεων της συνάρτησης.

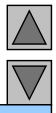


Outline

fig03_12.cpp (4 of 5)

```
72 // useGlobal modifies global variable x during each call
73 void useGlobal( void )
74 {
75    cout << endl << "global x is " << x
76         << " on entering useGlobal" << endl;
77    x *= 10;
78    cout << "global x is " << x
79         << " on exiting useGlobal" << endl;
80
81 } // end function useGlobal
```

Αυτή η συνάρτηση δε δηλώνει μεταβλητές. Χρησιμοποιεί την καθολική **x** που ορίζεται στην αρχή του προγράμματος.



Outline

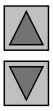
03_12.cpp
of 5)

03_12.cpp
tput (1 of 2)

```
local x in main's outer scope is 5
local x in main's inner scope is 7
local x in main's outer scope is 5

local x is 25 on entering useLocal
local x is 26 on exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal
```



```
local x is 25 on entering useLocal
local x is 26 on exiting useLocal

local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal

global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal

local x in main is 5
```



Outline

fig03_12.cpp
output (2 of 2)

Αναδρομή

- Αναδρομικές συναρτήσεις
  - Συναρτήσεις που καλούν τον εαυτό τους
  - Λύνουν μόνον μια βασική περίπτωση
- Εάν δεν συναντήσουμε τη βασική περίπτωση
  - Το πρόβλημα διασπάται σε μικρότερο(α) υποπρόβλημα(τα)
  - Καλείται ένα νέο αντίγραφο της συνάρτησης για να εργαστεί πάνω στο υποπρόβλημα (αναδρομική κλήση)
    - Η διαδικασία συγκλίνει προς τη βασική περίπτωση
    - Η συνάρτηση συνήθως καλεί τον εαυτό της εντός της εντολής return
  - Στο τέλος η βασική περίπτωση επιλύεται
    - Η λύση διαδίδεται προς τα πάνω, επιλύνοντας ολόκληρο το πρόβλημα

Αναδρομή

- Παράδειγμα: Παραγοντικοί αριθμοί

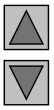
$$n! = n * (n - 1) * (n - 2) * \ldots * 1$$

  - Αναδρομική σχέση → n! = n * (n – 1)!

$$5! = 5 * 4!$$

$$4! = 4 * 3! \ldots$$

  - Βασική περίπτωση → 1! = 0! = 1

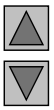


Outline

fig03_14.cpp (1 of 2)

```
1  // Fig. 3.14: fig03_14.cpp
2  // Recursive factorial function.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 unsigned long factorial( unsigned long ); // function prototype
13
14 int main()
15 {
16    // Loop 10 times. During each iteration, calculate
17    // factorial( i ) and display result.
18    for ( int i = 0; i <= 10; i++ )
19       cout << setw( 2 ) << i << "! = "
20            << factorial( i ) << endl;
21
22    return 0;  // indicates successful termination
23
24 } // end main
```

Data type **unsigned long** can hold an integer from 0 to 4 billion.



Outline

fig03_14.cpp (2 of 2)

```
25
26 // recursive definition of function fac
27 unsigned long factorial( unsigned long
28 {
29    // base case
30    if ( number <= 1 )
31       return 1;
32
33    // recursive step
34    else
35       return number * factorial( number - 1 );
36
37 } // end function factorial
```

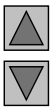
Η βασική περίπτωση εμφανίζεται όταν έχουμε **0!** ή **1!**. Όλες οι άλλες περιπτώσεις πρέπει να διασπαστούν (αναδρομικό βήμα).

fig03_14.cpp output (1 of 1)

```
 0! = 1
 1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
 7! = 5040
 8! = 40320
 9! = 362880
10! = 3628800
```

Συναρτήσεις με κενές λίστες παραμέτρων

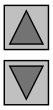
- Κενή λίστα παραμέτρων
  - **void** ή αφήνουμε τη λίστα παραμέτρων κενή
  - Η συνάρτηση **print** δεν έχει ορίσματα και δεν επιστρέφει καμιά τιμή
    - **void print();**
    - **void print(void);**



Outline

fig03_18.cpp
(1 of 2)

```
1  // Fig. 3.18: fig03_18.cpp
2  // Functions that take no arguments.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  void function1();        // function prototype
9  void function2( void );  // function prototype
10
11 int main()
12 {
13    function1();  // call function1 with no arguments
14    function2();  // call function2 with no arguments
15
16    return 0;     // indicates successful termination
17
18 } // end main
19
```



Outline

```
20 // function1 uses an empty parameter list to specify that
21 // the function receives no arguments
22 void function1()
23 {
24    cout << "function1 takes no arguments" << endl;
25
26 } // end function1
27
28 // function2 uses a void parameter list to specify that
29 // the function receives no arguments
30 void function2( void )
31 {
32    cout << "function2 also takes no arguments" << endl;
33
34 } // end function2
```

```
function1 takes no arguments
function2 also takes no arguments
```

fig03_18.cpp (2 of 2)

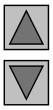
fig03_18.cpp output (1 of 1)

Inline Functions

- Inline functions
  - Χρειάζεται το Keyword **`inline`** πριν από τον ορισμό της συνάρτησης
  - Ζητά από τον μεταγλωττιστή να αντικαταστήσει κάθε κλήση της συνάρτησης στο πρόγραμμα μ' ένα αντίγραφο του κώδικα της συνάρτησης
    - Μειώνει την επιβάρυνση από την κλήση μιας συνάρτησης
    - Ο μεταγλωττιστής μπορεί να αγνοήσει την οδηγία **`inline`**
  - Καλή περίπτωση για μικρές και συχνά καλούμενες συναρτήσεις
- Παράδειγμα

```
inline double cube( const double s )
   { return s * s * s; }
```

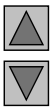
  - Το **`const`** μας λεει ότι η συνάρτηση δεν μπορεί να τροποποιήσει το **`s`**



Outline

fig03_19.cpp (1 of 2)

```
1  // Fig. 3.19: fig03_19.cpp
2  // Using an inline function to calculate.
3  // the volume of a cube.
4  #include <iostream>
5
6  using std::cout;
7  using std::cin;
8  using std::endl;
9
10 // Definition of inline function cube. Definition of function
11 // appears before function is called, so a function prototype
12 // is not required. First line of function definition acts as
13 // the prototype.
14 inline double cube( const double side )
15 {
16    return side * side * side;  // calculate cube
17
18 } // end function cube
19
```

Outline

fig03_19.cpp (2 of 2)

```
20 int main()
21 {
22    cout << "Enter the side length of your cube: ";
23
24    double sideValue;
25
26    cin >> sideValue;
27
28    // calculate cube of sideValue and display result
29    cout << "Volume of cube with side "
30         << sideValue << " is " << cube( sideValue ) << endl;
31
32    return 0;  // indicates successful termination
33
34 } // end main
```

fig03_19.cpp output (1 of 1)

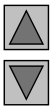
```
Enter the side length of your cube: 3.5
Volume of cube with side 3.5 is 42.875
```

Αναφορές Παραμέτρων

- Κλήση με τιμή (call by value)
  - Αντίγραφο της τιμής της μεταβλητής-όρισμα περνά στη συνάρτηση
  - Αλλαγές στο αντίγραφο δεν αλλάζουν την αρχική μεταβλητή
  - Εμποδίζονται ανεπιθύμητα side effects
- Κλήση με αναφορά (call by reference)
  - Η συνάρτηση μπορεί να προσπελάσει απ' ευθείας τα δεδομένα της μεταβλητής-όρισμα (δεν χρειάζεται αντίγραφο)
  - Οι αλλαγές επηρεάζουν και την αρχική μεταβλητή
- Call by **`const`** reference

Αναφορές Παραμέτρων

- Αναφορά παραμέτρου
  - Χρησιμοποιούμε το σύμβολο **`&`** μετά από τον τύπο δεδομένων στο πρωτότυπο
    - **`void myFunction( int &data )`**
    - Διαβάζεται "η **`data`** είναι μια αναφορά σ' έναν **`int`**"
  - Η κλήση της συνάρτησης παραμένει όπως γνωρίζουμε
    - Οι αλλαγές επηρεάζουν και την αρχική μεταβλητή

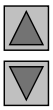


Outline

fig03_20.cpp (1 of 2)

```
1  // Fig. 3.20: fig03_20.cpp
2  // Comparing pass-by-value and pass-by-reference
3  // with references.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  int squareByValue( int );          // functi
10 void squareByReference( int & );  // function prototype
11
12 int main()
13 {
14    int x = 2;
15    int z = 4;
16
17    // demonstrate squareByValue
18    cout << "x = " << x << " before squareByValue\n";
19    cout << "Value returned by squareByValue: "
20         << squareByValue( x ) << endl;
21    cout << "x = " << x << " after squareByValue\n" << endl;
22
```

Προσέξτε τον τελεστή **&**, ο οποίος είναι ένδειξη του περάσματος με αναφορά.



Outline

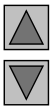
fig03_20.cpp (2 of 2)

```
23    // demonstrate squareByReference
24    cout << "z = " << z << " before squareByReference" << endl;
25    squareByReference( z );
26    cout << "z = " << z << " after squareByReference" << endl;
27
28    return 0;  // indicates successful termination
29 } // end main
30
31 // squareByValue multiplies number by itsel
32 // result in number and returns the new val
33 int squareByValue( int number )
34 {
35    return number *= number;  // caller's argument not modified
36
37 } // end function squareByValue
38
39 // squareByReference multiplies numberRef by its
40 // stores the result in the variable to which nu
41 // refers in function main
42 void squareByReference( int &numberRef )
43 {
44    numberRef *= numberRef;   // caller's argument modified
45
46 } // end function squareByReference
```

Changes **number**, but original parameter (**x**) is not modified.

Changes **numberRef**, an alias for the original parameter. Thus, **z** is changed.

```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```



Outline

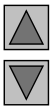
fig03_20.cpp
output (1 of 1)

Αναφορές Μεταβλητών

- Οι αναφορές μεταβλητών είναι aliases άλλων μεταβλητών
  - Ουσιαστικά πρόκειται για την ίδια μεταβλητή
  - Μπορεί να χρησιμοποιηθεί και εντός μιας συνάρτησης

```
int count = 1; // declare integer variable count
int &cRef = count; // create cRef as an alias for count
++cRef; // increment count (using its alias)
```

- Οι αναφορές πρέπει να αρχικοποιούνται όταν δηλώνονται
  - Διαφορετικά, ο μεταγλωττιστής παράγει error
  - Dangling reference
    - Αναφορά σε μη ορισμένη μεταβλητή



Outline

fig03_21.cpp (1 of 1)

fig03_21.cpp output (1 of 1)

```
1  // Fig. 3.21: fig03_21.cpp
2  // References must be initialized.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int x = 3;
11
12    // y refers to (is an alias for) x
13    int &y = x;
14
15    cout << "x = " << x << endl << "y = " << y << endl;
16    y = 7;
17    cout << "x = " << x << endl << "y = " << y << endl;
18
19    return 0;  // indicates successful termination
20
21 } // end main
```

y declared as a reference to **x**.

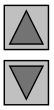
```
x = 3
y = 3
x = 7
y = 7
```


Προκαθορισμένα Ορίσματα

- Επιτρέπουν την κλήση μιας συνάρτησης χωρίς όλες τις παραμέτρους της
  - Αν δεν παρέχονται όλες οι παράμετροι, οι δεξιότερες που λείπουν παίρνουν τις προκαθορισμένες τιμές
  - Προκαθορισμένες τιμές
    - Μπορεί να είναι σταθερές, global μεταβλητές, ή function calls
- Θα πρέπει να οριστούν οι προκαθορισμένες τιμές στο πρωτότυπο της συνάρτησης

```
int myFunction( int x = 1, int y = 2, int z = 3 );
```

  - **myFunction(3)**
    - **x** = 3, **y** και **z** get defaults (rightmost)
  - **myFunction(3, 5)**
    - **x** = 3, **y** = 5 και **z** gets default



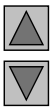
Outline

fig03_23.cpp (1 of 2)

```
1  // Fig. 3.23: fig03_23.cpp
2  // Using default arguments.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function prototype that specifies d
9  int boxVolume( int length = 1, int width = 1, int height = 1 );
10
11 int main()
12 {
13    // no arguments--use default values for all dimensions
14    cout << "The default box volume is: " << boxVolume();
15
16    // specify length; default width and height
17    cout << "\n\nThe volume of a box with length 10,\n"
18         << "width 1 and height 1 is: " << boxVolume( 10 );
19
20    // specify length and width; default height
21    cout << "\n\nThe volume of a box with length 10,\n"
22         << "width 5 and height 1 is: " << boxVolume( 10, 5 );
23
```

Θέτει τις προκαθορισμένες τιμές στο πρωτότυπο της συνάρτησης.

Κλήσεις συνάρτησης με κάποιες από τις παραμέτρους να λείπουν– οι δεξιότερες παράμετροι λαμβάνουν τις προκαθορισμένες τιμές.



Outline

fig03_23.cpp (2 of 2)

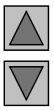
```
24    // specify all arguments
25    cout << "\n\nThe volume of a box with length 10,\n"
26         << "width 5 and height 2 is: " << boxVolume( 10, 5, 2 )
27         << endl;
28
29    return 0;  // indicates successful termination
30
31 } // end main
32
33 // function boxVolume calculates the volume of a box
34 int boxVolume( int length, int width, int height )
35 {
36    return length * width * height;
37
38 } // end function boxVolume
```

fig03_23.cpp output (1 of 1)

```
The default box volume is: 1

The volume of a box with length 10,
width 1 and height 1 is: 10

The volume of a box with length 10,
width 5 and height 1 is: 50

The volume of a box with length 10,
width 5 and height 2 is: 100
```

Μοναδιαίος τελεστής Scope Resolution ::

- Μοναδιαίος τελεστής Scope Resolution (: :)
  - Επιτρέπει την προσπέλαση μιας global μεταβλητής εάν μια τοπική μεταβλητή έχει το ίδιο όνομα.
  - Δεν χρειάζεται εάν τα ονόματα είναι διαφορετικά
  - Χρήση **::variable**
    - **y = ::x + 3;**
  - Είναι καλό να αποφεύγεται οι καθολικές μεταβλητές να έχουν το ίδιο όνομα με τις τοπικές.



Outline

fig03_24.cpp (1 of 2)

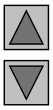
```
1  // Fig. 3.24: fig03_24.cpp
2  // Using the unary scope resolution operator.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setprecision;
11
12 // define global constant PI
13 const double PI = 3.14159265358979;
14
15 int main()
16 {
17    // define local constant PI
18    const float PI = static_cast< float >( ::PI );
19
20    // display values of local and global PI constants
21    cout << setprecision( 20 )
22         << "  Local float value of PI = " << PI
23         << "\nGlobal double value of PI = " << ::PI << endl;
24
25    return 0;  // indicates successful termination
```

Πρόσβαση του καθολικού **PI** με **::PI**.

Μετατροπή του καθολικού **PI** σε **float** για το τοπικό **PI**. Αυτό το παράδειγμα θα δείξει τις διαφορές μεταξύ **float** και **double**.

```
26
27 } // end main
```

```
Borland C++ command-line compiler output:
  Local float value of PI = 3.141592741012573242
Global double value of PI = 3.141592653589790007

Microsoft Visual C++ compiler output:
  Local float value of PI = 3.1415927410125732
Global double value of PI = 3.14159265358979
```



Outline

fig03_24.cpp (2 of 2)

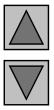
fig03_24.cpp output (1 of 1)

Υπερφόρτωση Συναρτήσεων (Function Overloading)

- Υπερφόρτωση Συναρτήσεων
  - Συναρτήσεις με το ίδιο όνομα και διαφορετικές παραμέτρους
  - Υλοποιούν παρόμοια δουλειά
    - Π.χ. συνάρτηση square για **int**s και συνάρτηση square για **float**s

```
int square( int x) {return x * x;}
float square(float x) { return x * x; }
```

- Οι συναρτήσεις που είναι υπερφορτωμένες διαχωρίζονται από την υπογραφή τους
  - Με βάση το όνομα και τους τύπους των παραμέτρων (η σειρά παίζει ρόλο)
  - Η αλλαγή απλά στον τύπο του επιστρεφόμενου αποτελέσματος δεν είναι υπερφόρτωση!!! (αλλά λάθος μεταγλώττισης)

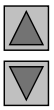


Outline

fig03_25.cpp (1 of 2)

```
1  // Fig. 3.25: fig03_25.cpp
2  // Using overloaded functions.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function square for int values
9  int square( int x )
10 {
11    cout << "Called square with int argument: " << x << endl;
12    return x * x;
13
14 } // end int version of function square
15
16 // function square for double values
17 double square( double y )
18 {
19    cout << "Called square with double argument: " << y << endl;
20    return y * y;
21
22 } // end double version of function square
23
```

Οι υπερφορτωμένες συναρτήσεις έχουν το ίδιο όνομα αλλά ξεχωρίζουν από τις διαφορετικές παραμέτρους.



Outline

fig03_25.cpp (2 of 2)

```
24 int main()
25 {
26    int intResult = square( 7 );         // calls int version
27    double doubleResult = square( 7.5 ); // calls double version
28
29    cout << "\nThe square of intege
30         << "\nThe square of double
31         << endl;
32
33    return 0;  // indicates successful termination
34
35 } // end main
```

Η κατάλληλη συνάρτηση καλείται με βάση το όρισμα (**int** ή **double**).

fig03_25.cpp output (1 of 1)

```
Called square with int argument: 7
Called square with double argument: 7.5

The square of integer 7 is 49
The square of double 7.5 is 56.25
```

Πρότυπα Συναρτήσεων (Function Templates)

- Συμπαγής τρόπος για να περιγράψουμε την υπερφόρτωση συναρτήσεων
  - Παράγεται μια διαφορετική συνάρτηση για διαφορετικούς τύπους δεδομένων
- Σύνταξη
  - Ο ορισμός ξεκινά με το keyword **`template`**
  - Παραμετρικοί τύποι δίνονται σε brackets **<>**
    - Κάθε παραμετρικός τύπος που ορίζεται παίρνει ως πρόθεμα **`typename`** ή **`class`** (synonyms)
    - Οι παραμετρικοί τύποι είναι placeholders για built-in types (π.χ. **`int`**) ή user-defined types
    - Χαρακτηρίζουν τον τύπο των ορισμάτων της συνάρτησης, επιστρεφόμενης τιμής ή τοπικών μεταβλητών
  - Ο ορισμός της συνάρτησης γίνεται όπως συνήθως με τη διαφορά ότι τώρα χρησιμοποιούμε αντί για τύπους δεδομένων τους παραμετρικούς τύπους

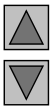
Πρότυπα Συναρτήσεων (Function Templates)

- Παράδειγμα

```
template < class T > // or template< typename T >
T square( T value1 )
{
   return value1 * value1;
}
```

  - **`T`** είναι ένας παραμετρικός τύπος
    - Ο τύπος της επιστρεφόμενης τιμής είναι ο ίδιος με τον τύπο του ορίσματος της συνάρτησης
  - Στην κλήση της συνάρτησης, το T αντικαθίσταται μ' έναν πραγματικό τύπο
    - Εάν βάλουμε **`int`**, όλα τα **`T`** γίνονται **`int`**

```
int x;
int y = square(x);
```

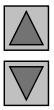
Outline

fig03_27.cpp (1 of 3)

```
1  // Fig. 3.27: fig03_27.cpp
2  // Using a function template.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // definition of function templat
10 template < class T >  // or template < typename T >
11 T maximum( T value1, T value2, T value3 )
12 {
13    T max = value1;
14
15    if ( value2 > max )
16       max = value2;
17
18    if ( value3 > max )
19       max = value3;
20
21    return max;
22
23 } // end function template maximum
24
```

Formal type parameter **T** placeholder for type of data to be tested by **maximum**.

maximum expects all parameters to be of the same type.

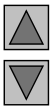


Outline

fig03_27.cpp (2 of 3)

```
25 int main()
26 {
27    // demonstrate maximum with int values
28    int int1, int2, int3;
29
30    cout << "Input three integer values: ";
31    cin >> int1 >> int2 >> int3;
32
33    // invoke int version of maximum
34    cout << "The maximum integer value is: "
35         << maximum( int1, int2, int3 );
36
37    // demonstrate maximum with double values
38    double double1, double2, double3;
39
40    cout << "\n\nInput three double values: ";
41    cin >> double1 >> double2 >> double3;
42
43    // invoke double version of maximum
44    cout << "The maximum double value is: "
45         << maximum( double1, double2, double3 );
46
```

maximum called with various data types.



Outline

```
47    // demonstrate maximum with char values
48    char char1, char2, char3;
49
50    cout << "\n\nInput three characters: ";
51    cin >> char1 >> char2 >> char3;
52
53    // invoke char version of maximum
54    cout << "The maximum character value is: "
55         << maximum( char1, char2, char3 )
56         << endl;
57
58    return 0;  // indicates successful termination
59
60 } // end main
```

fig03_27.cpp (3 of 3)

fig03_27.cpp output (1 of 1)

```
Input three integer values: 1 2 3
The maximum integer value is: 3

Input three double values: 3.3 2.2 1.1
The maximum double value is: 3.3

Input three characters: A C B
The maximum character value is: C
```


Άσκηση-1

- Τι είναι λάθος στο παρακάτω πρόγραμμα?
- Εάν δεν υπάρχει λάθος τι κάνει το πρόγραμμα?
- Το **EOF** δίνεται στην είσοδο ως **^Z** (DOS-παράθυρο) ή ως **^D** (Unix)

```
1  // ex03_49.cpp
2  #include <iostream>
3
4  using std::cin;
5  using std::cout;
6
7  int main()
8  {
9     int c;
10
11    if ( ( c = cin.get() ) != EOF ) {
12       main();
13       cout << c;
14    }
15
16    return 0;
17 }
```

Άσκηση-2

- Τι κάνει το παρακάτω πρόγραμμα?

```
1  // ex03_50.cpp
2  #include <iostream>
3
4  using std::cout;
5  using std::cin;
6  using std::endl;
7
8  int mystery( int, int );
9
10 int main()
11 {
12    int x, y;
13
14    cout << "Enter two integers: ";
15    cin >> x >> y;
16    cout << "The result is " << mystery( x, y ) << endl;
17    return 0;
18 }
19
20 // Parameter b must be a positive
21 // integer to prevent infinite recursion
22 int mystery( int a, int b )
23 {
24    if ( b == 1 )
25       return a;
26    else
27       return a + mystery( a, b - 1 );
28 }
```


Απάντηση-1

- Η Standard C++ δεν επιτρέπει αναδρομικές κλήσεις της **`main`**
- Σε υλοποιήσεις της C++ που αυτό επιτρέπεται (non-standard C++ compilers) το πρόγραμμα θα τυπώσει τους χαρακτήρες εισόδου σε αντίστροφη σειρά

Απάντηση-2

- Δοθέντων δύο ακεραίων αριθμών το πρόγραμμα τυπώνει το γινόμενό τους.