Συναρτήσεις

Functional programming

- Είναι η τεχνική προγραμματισμού κατά την οποία ένα «μεγάλο» και δύσκολο πρόβλημα το «σπάμε» σε μικρότερα και απλούστερα
- Τα μικρότερα προβλήματα είναι πιο εύκολο να δημιουργηθούν απευθείας σε κώδικα
- Γενικά οι functions (συναρτήσεις) κάνουν ένα πρόβλημα πιο εύκολο
  - Στην κατανόηση
  - Στην δημιουργία / μεταβολή
  - Στον έλεγχο / debug
  - Στην ανάπτυξη από ομάδες

Έτοιμες συναρτήσεις

- Στην C++ υπάρχουν βιβλιοθήκες με έτοιμες functions τις οποίες μπορούμε να χρησιμοποιήσουμε εκτελώντας ένα ***function call***

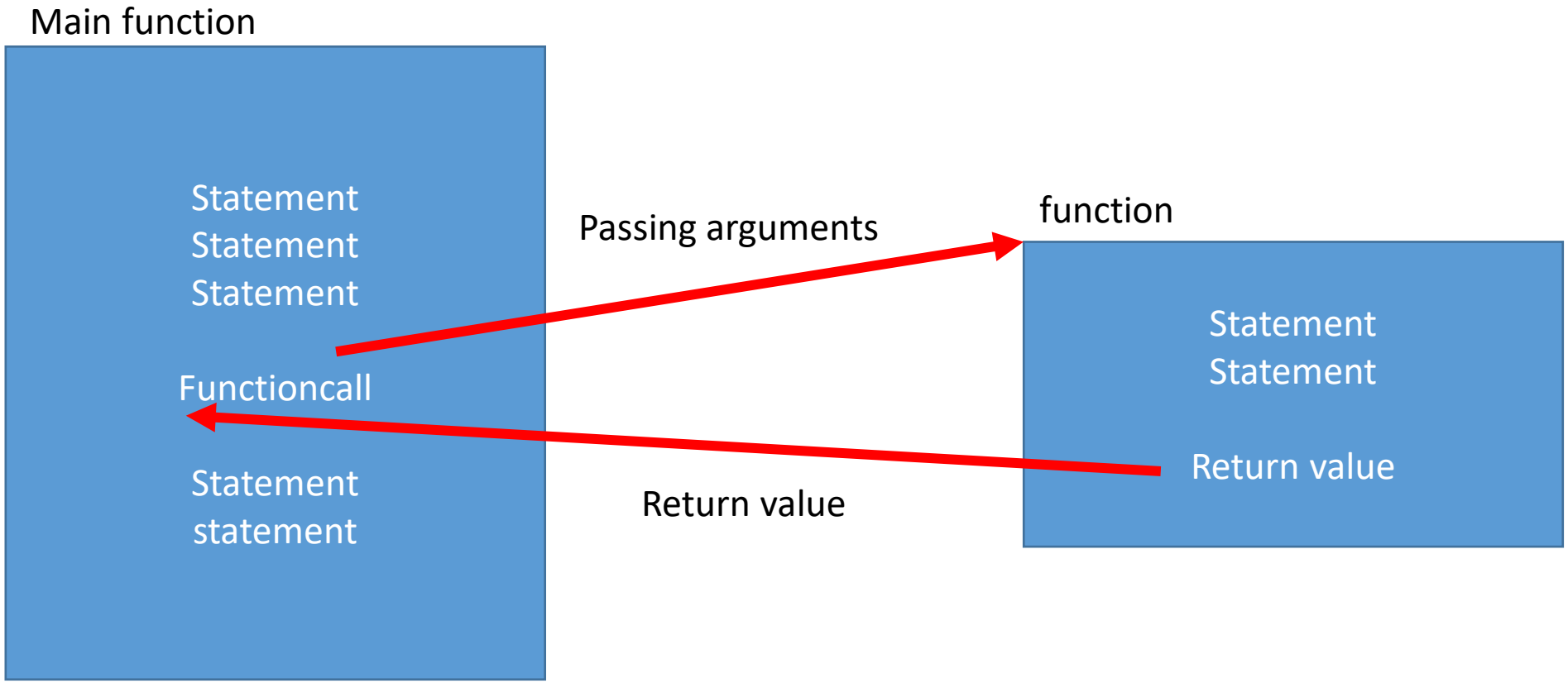
the_root = sqrt(9.0)

Μεταβλητή που θα αποθηκεύσει την τιμή που επιστρέφει η συνάρτηση

Όνομα συνάρτησης που καλούμε

Argument – όρισμα συνάρτησης
Αρχική τιμή που θα αποδοθεί στην παράμετρο της συνάρτησης

Διάγραμμα λειτουργίας ενός function call



Μερικές παρατηρήσεις για τα function call

Υπάρχουν οι παρακάτω περιπτώσεις:

- Ένα function call να έχει παραπάνω από ένα ορίσματα ή να μην έχει καθόλου
- Στην θέση των ορισμάτων να έχουμε ένα expression
- Το function call να χρησιμοποιηθεί σε ένα expression
- Μπορεί ένα function call να μην επιστρέφει τιμή

```
final = pow(3.4, 8);
tmp = rand();

final = sqrt(8*tmp/4);

final = (4* sqrt(8))/3;
tmp = sqrt(rand());
```

Function libraries

- Οι έτοιμες συναρτήσεις που παρέχει η C++ βρίσκονται οργανωμένες μέσα σε Libraries
- Για να μπορέσουμε να χρησιμοποιήσουμε την συνάρτηση θα πρέπει να κάνουμε include την library
- Για παράδειγμα για να χρησιμοποιήσουμε τις μαθηματικές συναρτήσεις κάνουμε include την βιβλιοθήκη cmath

#include <cmath>

- Για την χρήση των σταθερών (π.χ. M_PI) χρειάζεται στην αρχή

#define _USE_MATH_DEFINES

Μερικές έτοιμες συναρτήσεις

- **abs(x)** παίρνει όρισμα ακέραιο επιστρέφει απόλυτη τιμή ακέραιο
- **fabs(x)** παίρνει όρισμα double επιστρέφει απόλυτη τιμή double
- **rand()** επιστρέφει έναν τυχαίο ακέραιο που είναι από μηδέν μέχρι RANDMAX (χρειάζεται srand για seeding)

Συναρτήσεις που δημιουργούνται από τον χρήστη

Παράδειγμα

- Έστω ότι θέλουμε να δημιουργήσουμε μια συνάρτηση που να λέγεται **calc_circle**
- Η συνάρτηση αυτή θα παίρνει δύο ορίσματα εισόδου
  - Έναν ακέραιο που αν έχει τιμή 1 θα επιστρέφει την περίμετρο του κύκλου αν έχει την τιμή 2 θα επιστρέφει τον όγκο του κύκλου και για οποιαδήποτε τιμή θα επιστρέφει το -1
  - Έναν πραγματικό διπλής ακρίβειας που θα έχει την ακτίνα του κύκλου για τον οποίο θέλουμε να υπολογίσουμε την περίμετρο ή τον όγκο

Δήλωση - ορισμός

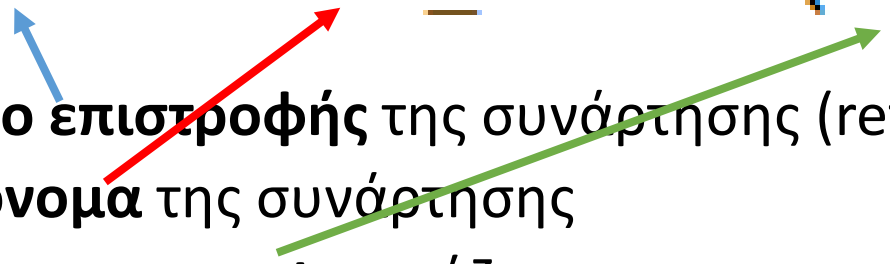
Για την δημιουργία μιας συνάρτησης από τον χρήστη χρειάζονται δύο ενέργειες:

- Η δημιουργία μιας **δήλωσης (declaration)** της συνάρτησης. Πρέπει να υπάρχει πριν από το function call. Στην ουσία ενημερώνει τον compiler ότι υπάρχει αυτή η συνάρτηση
- Η δημιουργία του **ορισμού (definition)** της συνάρτησης που περιέχει και τον κώδικα που θα εκτελέσει όταν καλεστεί.

Δήλωση - declaration

- Η δήλωση μιας συνάρτησης εμφανίζει

```
double calc_circle(int c, double r);
```



  - **Τύπο επιστροφής** της συνάρτησης (return type)
  - Το **όνομα** της συνάρτησης
  - Πόσα **argument** χρειάζεται
  - Ποιος είναι ο **τύπος του κάθε argument**
  - Ποια είναι τα **formal parameter names** (είναι οι μεταβλητές της συνάρτησης που θα «κρατήσουν» τις τιμές των arguments όταν γίνει το function call)

Παρατηρήσεις για την δήλωση

- Πρέπει να υπάρχουν πάντα οι παρενθέσεις ακόμα και αν η συνάρτηση δεν χρειάζεται arguments
- Ακόμα και αν τα arguments είναι του ίδιου τύπου πρέπει πριν από το κάθε ένα να γράφουμε τον τύπο του.
- Στην δήλωση μιας συνάρτησης μπορούμε να παραβλέψουμε το όνομα των formal parameters και να γράψουμε μόνο τον τύπο

Ορισμός definition

- Περιέχει τα ίδια στοιχεία με το declaration (προσοχή ΠΡΕΠΕΙ να υπάρχουν τα formal parameter names)
- Επιπρόσθετα περιέχει και τον κώδικα που δείχνει το τι κάνει αυτή η συνάρτηση
- Περιέχει την εντολή return

```
double calc_circle(int c, double r)
{
    if (c == 1)
        return 2 * M_PI * r;
    else if (c == 2)
        return M_PI * pow(r, 2);
    else
        return -1;
    // never reach here
}
```

Return statement

- Οι συναρτήσεις στην C++ επιστρέφουν (με τον κανονικό τρόπο) **μόνο ΜΙΑ τιμή με την χρήση της return**
  - Η τιμή αυτή πρέπει να είναι ίδιου τύπου με τον τύπο επιστροφής της συνάρτησης
  - Αν ο τύπος επιστροφής είναι ο void τότε δεν επιστρέφουν τιμή
- Όταν ο κώδικας της συνάρτησης εκτελέσει την εντολή return τότε σταματά η εκτέλεση του
- Μπορεί να επιστρέφει και κάποιο expression (ίδιου τύπου)
- Όλοι οι πιθανοί τρόποι τερματισμού της function πρέπει να περιέχουν return

Παράδειγμα συνάρτησης 2

- Να δημιουργήσετε μία συνάρτηση που να δέχεται σαν ορίσματα δύο αριθμούς τύπου double. Ο πρώτος θα είναι η ακτίνα της βάσης ενός κώνου και ο δεύτερος το ύψος του.
- Η συνάρτηση θα υπολογίζει και θα επιστρέφει τον όγκο του κώνου όπως υπολογίζεται από την μαθηματική σχέση

$$V = \frac{1}{3}\pi r^2 h$$

Παράδειγμα function

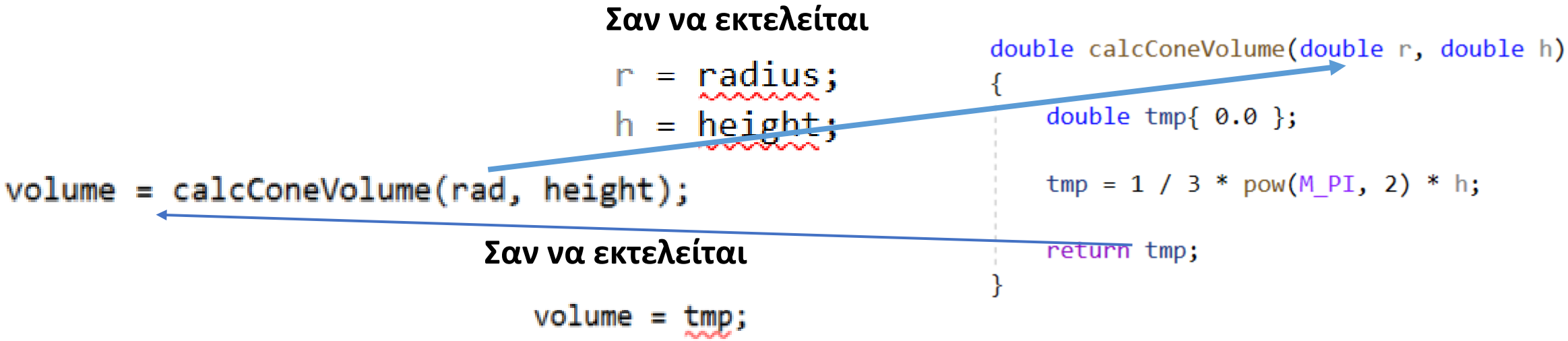
```
int main() {
    double radius{ 0.0 }, height{ 0.0 };
    double volume{ 0.0 };

    cout << "Dose aktina, ypsos conou: ";
    cin >> radius >> height;
    volume = calcConeVolume(radius, height);

    cout << "Cone aktina:" << radius << " ipsos:" << height;
    cout << " => Ogkos:" << volume << endl;
}
```

```
double calcConeVolume(double r, double h)
{
    double tmp{ 0.0 };

    tmp = 1 / 3 * pow(M_PI, 2) * h;

    return tmp;
}
```

Function call

«Κρυφές» αναθέσεις σε function call

- Κάθε φορά που γίνεται κλήση μιας συνάρτησης έχουμε δύο «κρυφές» αναθέσεις
  1. Η ανάθεση των τιμών των arguments στα formal parameters
  2. Η ανάθεση της τιμής επιστροφής στην μεταβλητή που θα την αποθηκεύσει



Συμφωνία μεταξύ function call - header

- Πρέπει ***ο αριθμός και ο τύπος των formal parameters*** που υπάρχουν στην δήλωση μιας συνάρτησης να ταιριάζουν με τον αριθμό και τον τύπο των arguments στο function call
  - ΠΡΟΣΟΧΗ μπορεί όμως να γίνει η αυτόματη μετατροπή τύπων
- **Η σειρά** με την οποία περιγράφει η δήλωση μιας συνάρτησης για τα formal parameters πρέπει να ταιριάζει με την σειρά με την οποία δίνονται τα arguments στο function call
  - Π.χ. η προηγούμενη συνάρτηση περιμένει πρώτα την τιμή της ακτίνας και μετά την τιμή του ύψους. ΔΕΝ μπορεί το function call να τα δίνει ανάποδα!!!

Procedural abstraction

- Είναι η τεχνική με την οποία σχεδιάζουμε μία συνάρτηση με τέτοιο τρόπο κάποιος να γνωρίζει ΜΟΝΟ πως να την χρησιμοποιήσει
  - Δηλαδή τι να δώσει σαν είσοδο και τι επιστρέφει σαν έξοδο
  - Ουσιαστικά δηλαδή την ΔΗΛΩΣΗ (declaration) της συνάρτησης
- ΔΕΝ ενδιαφέρει το πως υπολογίζει αυτό που επιστρέφει
- Είναι το προγραμματιστικό ανάλογο ενός «μαύρου κουτιού»

Συναρτήσεις που δεν επιστρέφουν τίποτα

Συναρτήσεις void

- Οι συναρτήσεις που δεν επιστρέφουν τίποτα διαφέρουν σε δύο σημεία από τις υπόλοιπες συναρτήσεις

```
void myfunc(int a, int b);
```

1. Σαν τύπος επιστροφής τους δηλώνεται το void
2. Η εντολή return δεν περιέχει κάποιο expression όταν τελειώσει κώδικας τελειώνει και η συνάρτησ

```
void myfunc(int a, int b) {
    int tmp;

    tmp = pow(a, b);
    cout << "a se dynami b einai " << tmp << endl;
    return;
}
```

- Η συνάρτηση main επιστρέφει INT αλλά μπορούμε να παραβλέψουμε το return

Function call σε void functions

- Επειδή οι void functions δεν επιστρέφουν τιμή ΔΕΝ μπορούν να χρησιμοποιηθούν
  - Σε expressions
  - Σε αναθέσεις
  - Σε cout
- Ουσιαστικά είναι από μόνες τους μία εντολή

```
tmp = myfunc(3, 4);
cout << myfunc(3, 4);
sqrt(myfunc(3, 4));
```

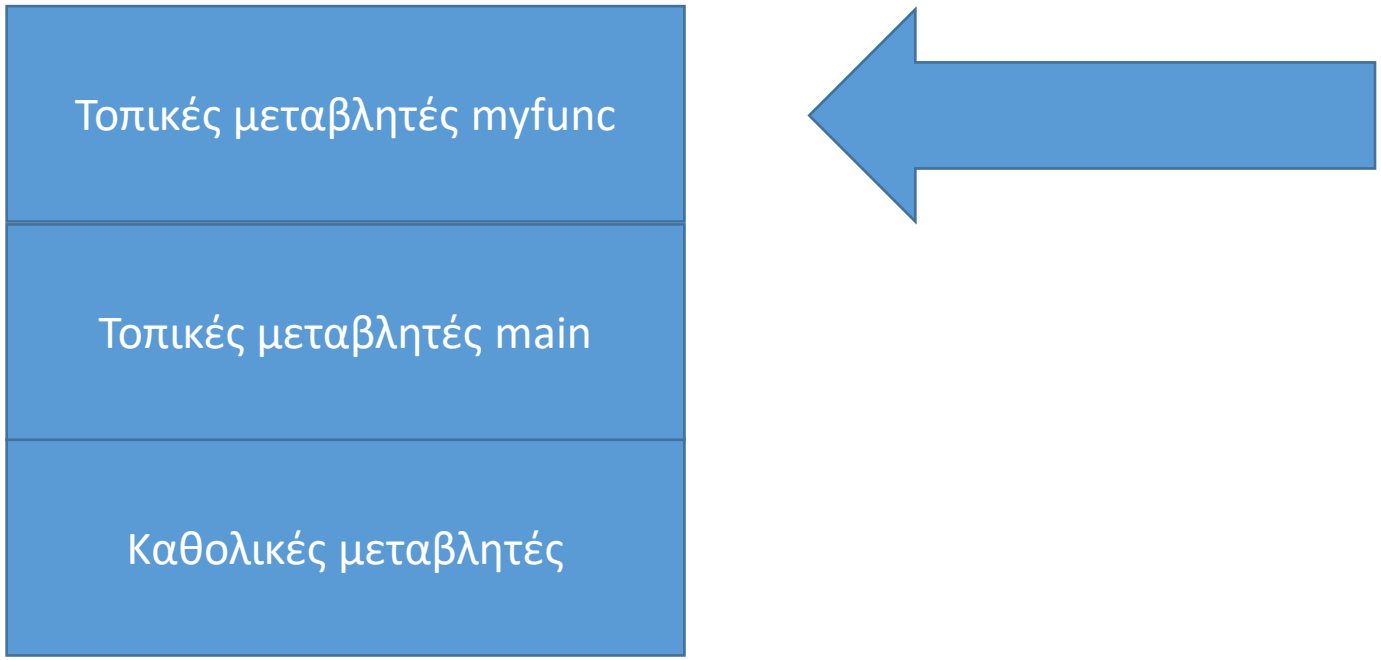
```
myfunc(3, 4);
```

Τοπικές - αυτόματες μεταβλητές

Τοπικές – αυτόματες μεταβλητές

- Οι μεταβλητές που δηλώνονται σε μία function ονομάζονται σαν τοπικές ή αυτόματες και:
  - Έχουν **εμβέλεια – scope** την συνάρτηση στην οποία ορίστηκαν
  - Έχουν **διάρκεια ζωής** – lifetime την συνάρτηση στην οποία ορίστηκαν (εκτός αν οριστούν σαν static). Όταν τελειώσει η συνάρτηση ο χώρος στην μνήμη στην οποία αποθηκεύονται ελευθερώνεται
  - Και **οι formal parameters είναι τοπικές – αυτόματες μεταβλητές**
- ΠΡΟΣΟΧΗ οι μεταβλητές που δηλώνονται στην main είναι τοπικές στην main.

Απλοποιημένη παράσταση του frame stack



Κάθε φορά έχει πρόσβαση μόνο στο κομμάτι μνήμης για την συνάρτηση καθώς και στο global

Παράδειγμα 3

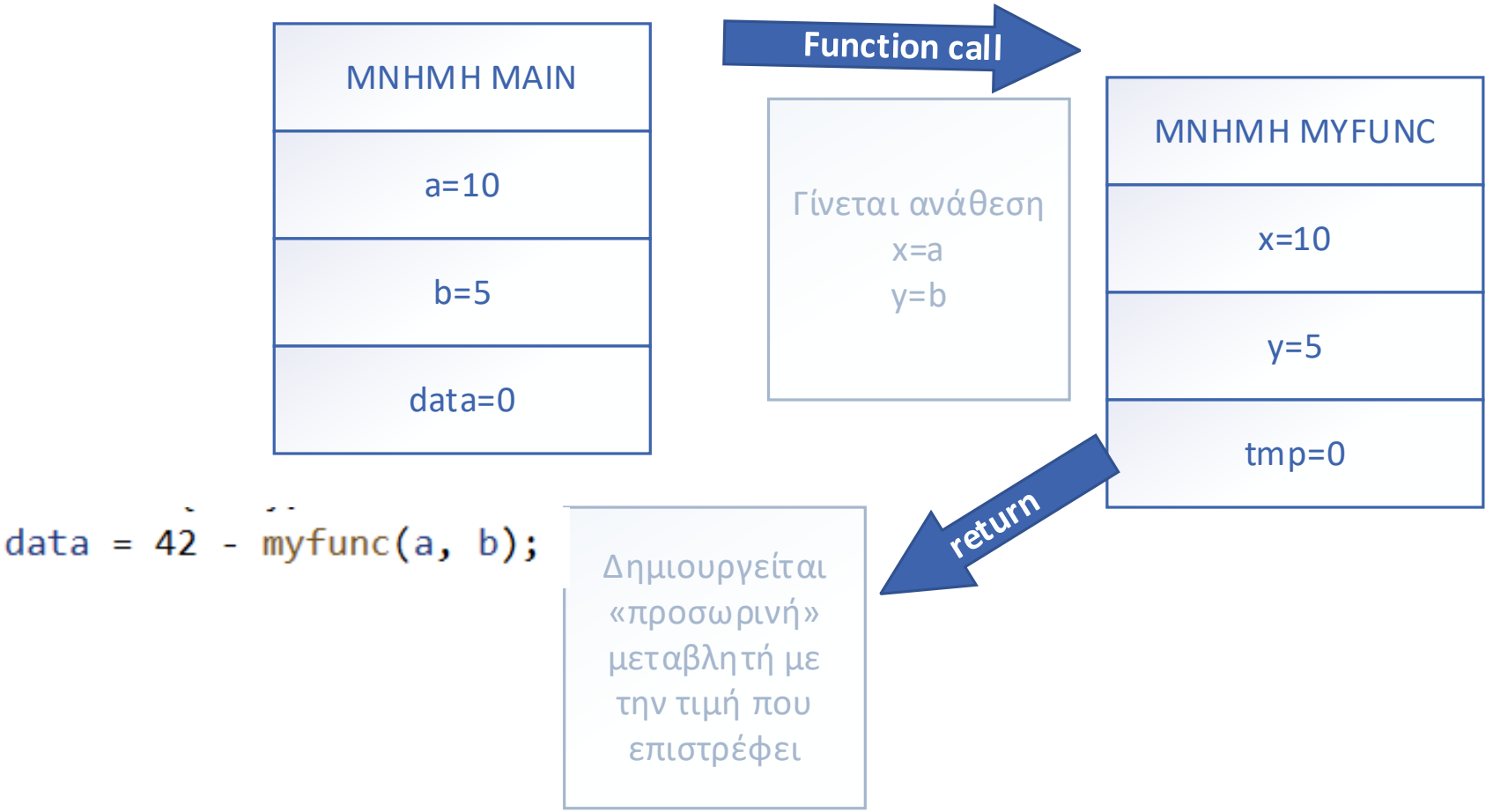
- Ένα απλό πρόγραμμα περιέχει
  - Main
  - Συνάρτηση με δύο int ορίσματα που επιστρέφει μία int τιμή
- Στην main ορίζονται 3 μεταβλητές a, b, data
- Στην συνάρτηση εκτός από τις formal parameters ορίζεται και μία μεταβλητή tmp
- Η κάθε μία συνάρτηση έχει την δική της μνήμη

Παράδειγμα 3

```
int main() {
    int a{ 10 }, b{ 5 };
    int data{ 0 };
    data = 42 - myfunc(a, b);
}
```

```
int myfunc(int x, int y)
{
    int ret{ 0 };

    ret = 2 * x - y;
    return ret;
}
```

Μία ματιά στην μνήμη




```
data = 42 - myfunc(a, b);
```

Μεταβλητές static

- Οι μεταβλητές έχουν την ίδια εμβέλεια και διάρκεια ζωής
- Οι static διαφέρουν
  - Εμβέλεια – στο scope που ορίστηκαν (πρέπει να αρχικοποιηθούν)
  - Διάρκεια ζωής – όλο το πρόγραμμα
- Στο διπλανό πρόγραμμα μετράει τον αριθμό κλήσεων

```
void add_one(void) {
    static int x{ 0 };

    ++x;
    cout << "func x="<<x << endl;
}
```

```
const double hour_sal = 6.2, extra_sal = 7.7;
double calc_wage(int h);
void display_sal(int emp_code, double w=100.0);

int main() {
    int emp_code = 0, hours = 0;
    double wage = 0.0;

    cout << "Dose kodiko ipalilou: ";
    cin >> emp_code;
    cout << "Poses ores doulepse: ";
    cin >> hours;

    wage = calc_wage(hours);
    display_sal(emp_code);
    return 0;
}
```

```
double calc_wage(int h) {
    if (h > 40) {
        return 40 * hour_sal + (h - 40)*extra_sal;
    }
    else
        return h * hour_sal;
}

void display_sal(int emp_code, double w) {
    cout << "The employer with code " << emp_code << endl;
    cout << "will get " << w << " euros this week\n";
}
```

default arguments

- Μπορούμε να δηλώσουμε μία formal parameter να έχει ένα default argument
  - Αυτό γίνεται αναθέτοντας μία τιμή σε αυτή την parameter στην δήλωση της συνάρτησης

```
void display_sal(int emp_code, double w=100.0);
```

- Δηλαδή να μπορεί στο function call να μην ορίζει τιμή για αυτή αλλά να παίρνει μια προκαθορισμένη. Για παράδειγμα

```
display_sal(emp_code);
```

Overloading

Function Overloading

- Πολλές φορές χρειαζόμαστε functions που κάνουν την ίδια δουλειά απλώς λειτουργούν με διαφορετικού τύπου η διαφορετικό αριθμό arguments
  - Π.χ. Να υπολογίζει κάτι με είσοδο ακέραιο και να κάνει την ίδια λειτουργία με είσοδο δεκαδικό
- Η C++ μας επιτρέπει να δηλώσουμε τις συναρτήσεις αυτές με το ίδιο όνομα.
- Δηλαδή function overloading είναι η τεχνική που επιτρέπει σε δύο συναρτήσεις να έχουν το ίδιο όνομα φτάνει να διαφέρουν στον αριθμό ή και στον τύπο των formal parameters

Function overloading

- Οι functions πρέπει να διαφέρουν
  - Είτε στον αριθμό των παραμέτρων
  - Είτε στον τύπο των παραμέτρων
- **ΔΕΝ** μπορούν να διαφέρουν ΜΟΝΟ στον τύπο επιστροφής

```
int my_func(int a);
int my_func(int a, int b);
int my_func(double a, double b);
```

```
double my_func(int a, int b);
```

Function call σε overloading

- Στο function call ο compiler ανάλογα με τον αριθμό και τον τύπο των arguments καλεί και την κατάλληλη συνάρτηση
  - ΠΡΟΣΟΧΗ μπορεί να γίνει και αυτόματη μετατροπή τύπων
  - Υπάρχουν περιπτώσεις όπου ΔΕΝ μπορεί να επιλέξει ανάμεσα στις συναρτήσεις

```
cout << my_func(3) << endl;
cout << my_func(3,5) << endl;
cout << my_func(3.4,6.5) << endl;

cout << my_func(3.4) << endl;
cout << my_func(3, 4.5) << endl;
```

Πέρασμα με αναφορά

Μεταβλητές με αναφορά

- Δηλώνουμε μια μεταβλητή σαν αναφορά σε μία άλλη μεταβλητή τοποθετώντας μπροστά από το όνομα της τον χαρακτήρα &
  - ΑΠΑΡΑΙΤΗΤΑ αρχικοποιείται με το όνομα μιας άλλης μεταβλητής
- Η αναφορά ΔΕΝ καταλαμβάνει χώρο στην μνήμη αλλά στην ουσία είναι «ψευδώνυμο» για μία άλλη

```
int main()
{
    int a = 9;
    int &b = a;

    cout << b;
    b = 11;
    cout << a;

    return 0;
}
```

Function call by value

- Όταν γίνεται ένα function call οι αναθέτονται οι τιμές των ορισμάτων στις formal parameters
- Αυτό είχε σαν αποτέλεσμα να ΜΗΝ μπορεί η συνάρτηση να μεταβάλει τις τιμές των ορισμάτων

```
int a = 2, b = 3;
int apot;

apot = my_func(a,b);
```

```
Values:
a = 2
b = 3
apot = 30
```

```
int my_func(int x, int y) {
    int tmp;
    x = 5;
    y = 6;
    tmp = x * y;
    return tmp;
}
```

Function call by reference

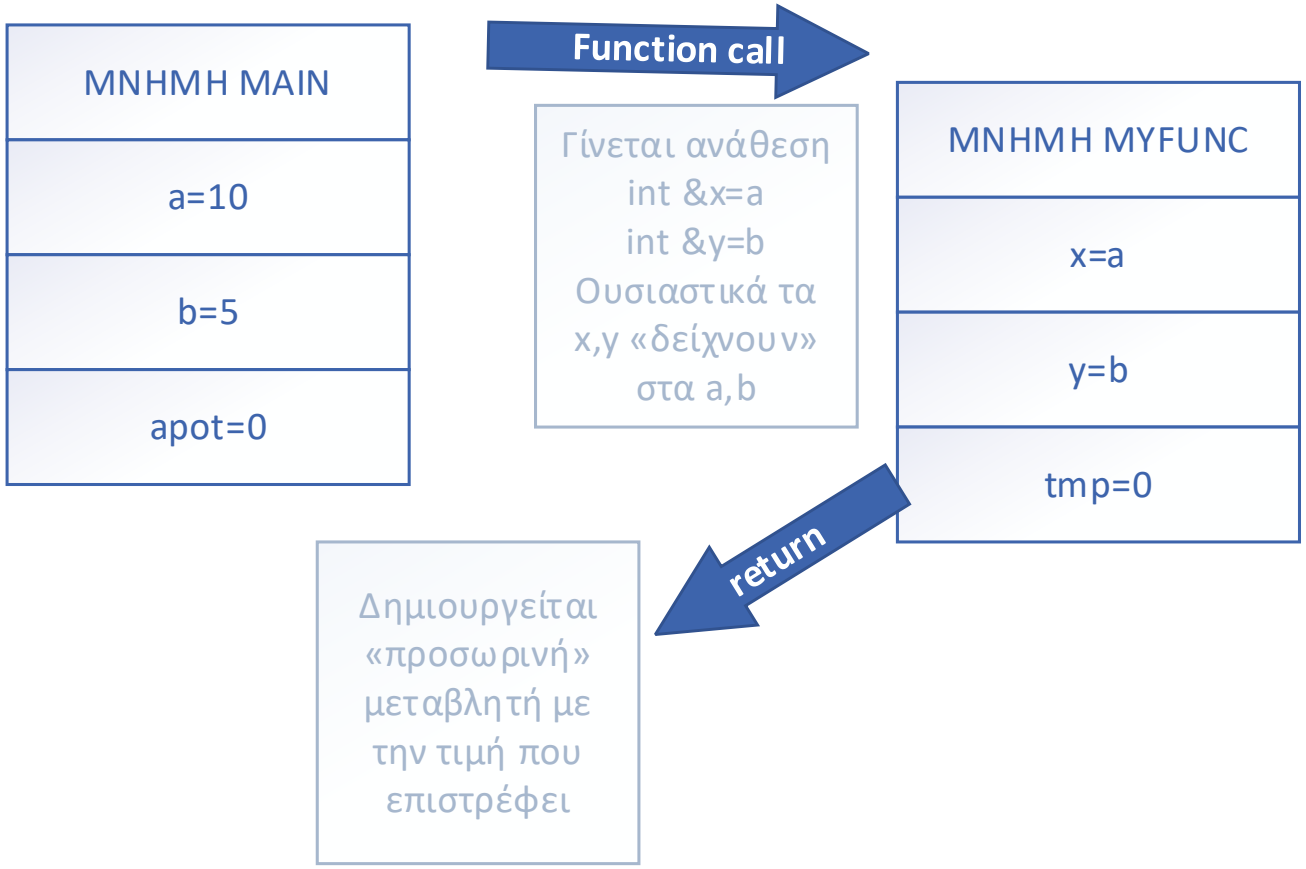
- Μπορούμε να ορίσουμε μία ή περισσότερες formal parameters σαν references
- Σε αυτή την περίπτωση τα ορίσματα που ανατίθενται σε αυτές μπορούν να μεταβληθούν και μέσω των formal parameters ΜΕΣΑ στην συνάρτηση

```
int a = 2, b = 3;
int apot;

apot = my_func(a,b);
```

```
Values:
a = 5
b = 6
apot = 30
```

```
int my_func(int &x, int &y) {
    int tmp;
    x = 5;
    y = 6;
    tmp = x * y;
    return tmp;
}
```

Μια ματιά στην μνήμη



Function call by reference

- Ουσιαστικά όποιο formal parameter μετατρέπουμε σε reference με αυτόν τον τρόπο μετατρέπουμε την αντίστοιχη είσοδο σε είσοδο/έξοδο της συνάρτησης
  - Αυτό δεν είναι απαραίτητα καλό
- Είναι μία λύση όταν θέλουμε μία συνάρτηση να επιστρέφει παραπάνω από μία τιμές

Function call by reference and const

- Πολλές φορές θέλουμε να έχουμε πέρασμα τιμών με reference χωρίς όμως να θέλουμε να μεταβάλουμε την τιμή τους
  - Για παράδειγμα όταν «περνάμε» σαν παραμέτρους μεγάλα σε μέγεθος αντικείμενα
- Σε αυτή την περίπτωση μπορούμε να συνδυάσουμε το reference με το πρόθεμα const
  - Με αυτό τον τρόπο και δεν γίνεται «αντίγραφο» του αντικειμένου αλλά και δεν υπάρχει η περίπτωση να μεταβληθεί η τιμή του από λάθος.
  - Η τεχνική αυτή χρησιμοποιείται πολύ στον OOP

Functions που επιστρέφουν reference

Συναρτήσεις που επιστρέφουν reference

- Οι συναρτήσεις που επιστρέφουν reference στην ουσία επιστρέφουν μια θέση μνήμης (αντικείμενο)
- Επομένως μπορούν να χρησιμοποιηθούν τόσο στο δεξί όσο ΚΑΙ στο αριστερό μέρος μιας εντολής ανάθεσης

```
int& func(int& x, int& y, int& z);
```

```
cout << "kalo func(a,b,c):" << func(a, b, c) << endl;
func(a, b, c) = 7;
```

Παράδειγμα

```
int main() {
    int a{ 2 }, b{ 4 }, c{ 3 };

    cout << "kalo func(a,b,c):" << func(a, b, c) << endl;
    func(a, b, c) = 7;
    func(a, b, c) = 9;

    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    cout << "c=" << c << endl;
}
```

```
int& func(int& x, int& y, int& z)
{
    if (x <= y && x <= z)
        return x;
    else if (y <= x && y <= z)
        return y;
    else
        return z;
}
```

Συναρτήσεις που επιστρέφουν reference

- ΠΡΟΣΟΧΗ
- Οι συναρτήσεις που επιστρέφουν reference θα πρέπει να επιστρέφουν αντικείμενα που υπάρχουν και μετά την συνάρτηση
- Επομένως ΔΕΝ μπορούν να επιστρέφουν αυτόματες μεταβλητές που δημιουργούνται μέσα στην συνάρτηση

```
int& func(int& x, int& y, int& z)
{
    return x * y * z;
}

int& func2()
{
    int data = 40;
    return data;
}
```