

# ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ 2025-2026

<https://helios.ntua.gr/course/view.php?id=869>

20/2/2026

Διδάσκοντες:

Νίκος Παπασπύρου, Καθ.

Βασίλης Βεσκούκης, Καθ.

Νίκος Λεονάρδος, Επ. Καθ.

Πέτρος Ποτίκας, ΕΔΙΠ

Γιώργος Σιόλας, ΕΔΙΠ

Σωτήρης Κοκόσης, ΕΔΙΠ

progtech@cslab.ece.ntua.gr

## Διαφάνειες παρουσιάσεων

- Η γλώσσα προγραμματισμού C++
- Αρχές αντικειμενοστρεφούς προγραμματισμού
- Δομές δεδομένων

# Σύντομη επανάληψη της C++

(i)

□ Γεια σου κόσμε!

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {  
    cout << "Hello world!" << endl;  
}
```

# Σύντομη επανάληψη της C++

(ii)

- Δώσε μου δύο αριθμούς...

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {  
    int a, b;  
    cout << "Give me two numbers: ";  
    cin >> a >> b;  
    cout << "You gave me " << a  
        << " and " << b << endl;  
}
```

# Σύντομη επανάληψη της C++

(iii)

□ ... και θα σου βρω το ΜΚΔ τους

```
int gcd(int x, int y) {
    while (x > 0 && y > 0)
        if (x > y) x %= y; else y %= x;
    return x + y;
}

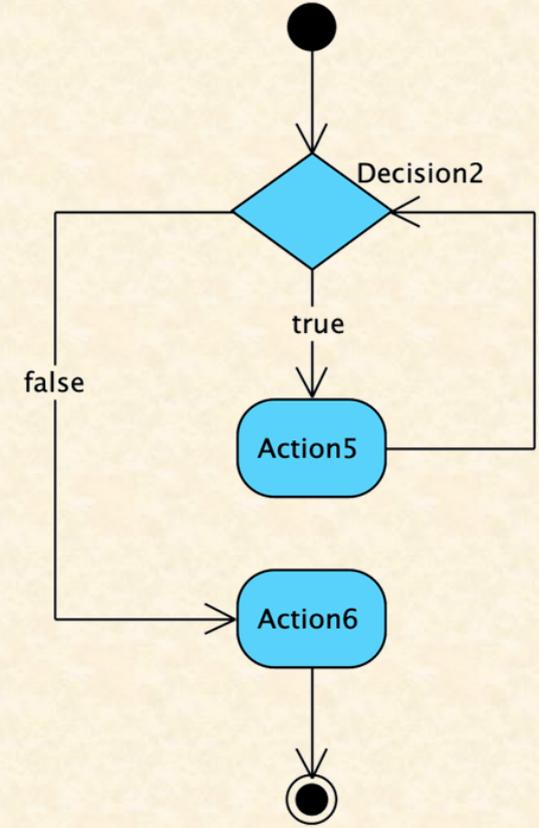
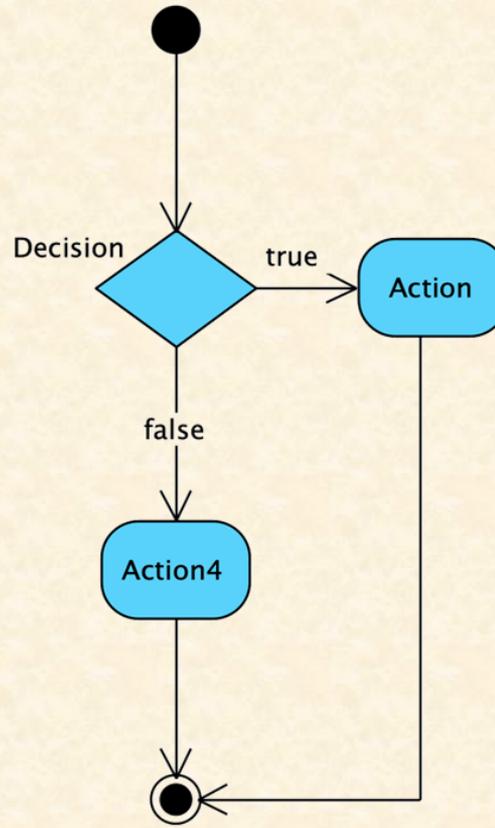
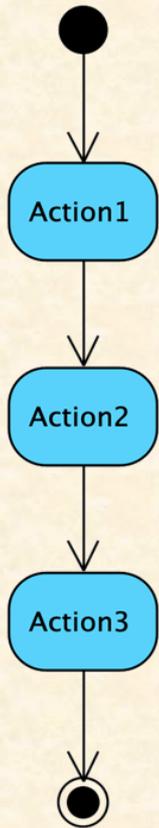
int main() {
    cout << "Give me two numbers: ";
    int a, b;
    cin >> a >> b;
    cout << "GCD (" << a << ", " << b
        << ") = " << gcd(a, b) << endl;
}
```

# Προγραμματισμός - επανάληψη

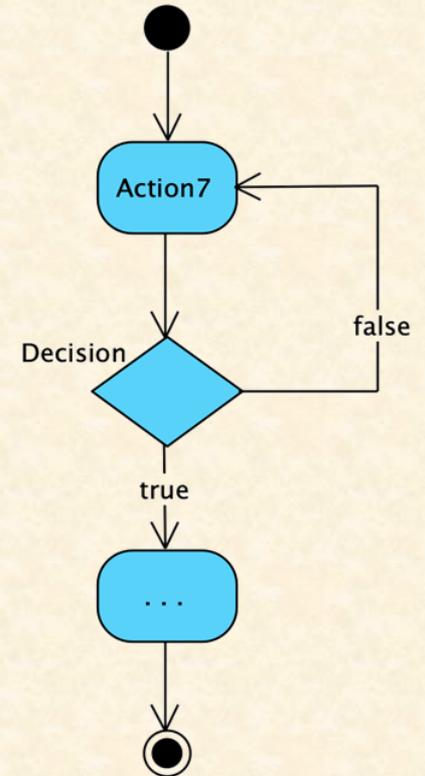
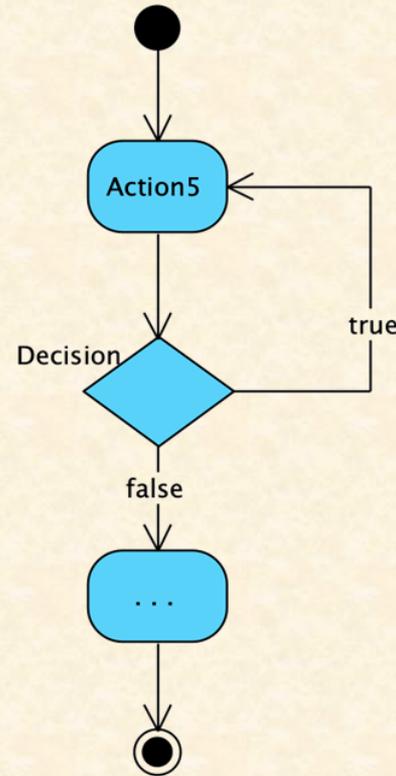
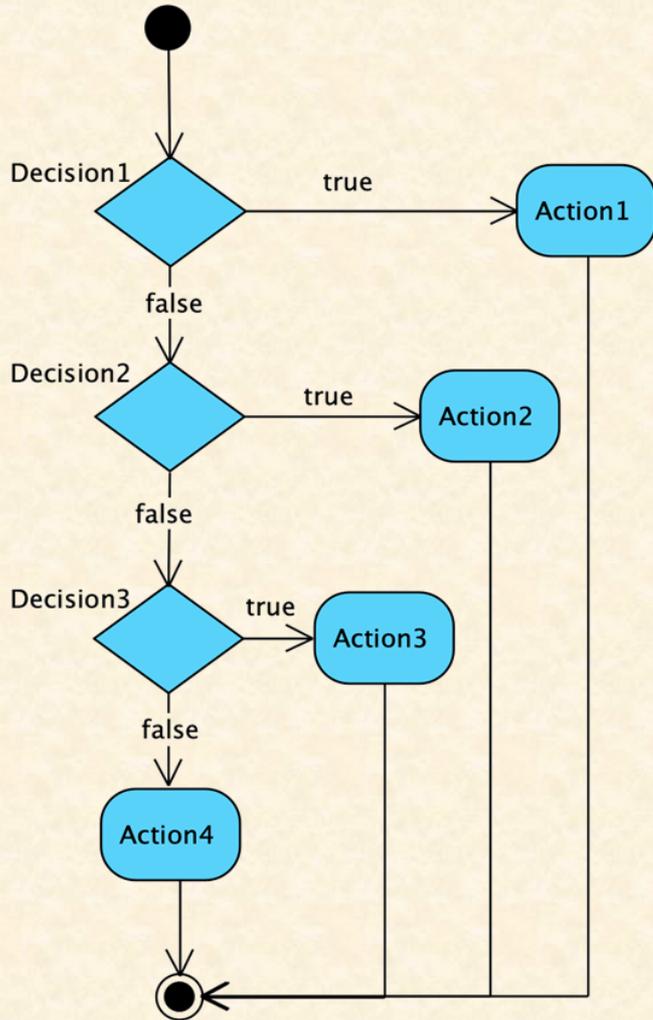
---

- **Algorithms + Data Structures = Programs**
  - **Δεδομένα**: τύποι, οργάνωση (δομές δεδομένων), λογική και φυσική υπόσταση
  - **Αλγόριθμοι**: τρόποι επίλυσης υπολογιστικών προβλημάτων που υλοποιούνται με εκφράσεις μιας γλώσσας προγραμματισμού: σειριακή, υπό συνθήκη, ή επαναληπτική εκτέλεση
- **Δομή προγράμματος**: συναρτήσεις, παράμετροι
- **Λοιπά**: μόνιμη αποθήκευση, δικαιώματα πρόσβασης, επικοινωνίες, ασφάλεια, κ.ά.

# Αλγόριθμοι: ροή ελέγχου εκτέλεσης



# Αλγόριθμοι: ροή ελέγχου εκτέλεσης



# Εύρεση αλγορίθμου για κάθε πρόβλημα

Γράψτε ένα πρόγραμμα το οποίο θα βρίσκει και θα εμφανίζει στην οθόνη όλους τους τετραψήφιους ακεραίους που είναι ίσοι με το άθροισμα των ψηφίων τους, υψωμένων στην τέταρτη δύναμη. Π.χ.  $1634 = 1^4 + 6^4 + 3^4 + 4^4$ .

Γράψτε ένα πρόγραμμα που θα ελέγχει αν δύο πίνακες  $N$  διαφορετικών ακεραίων αριθμών  $A$  και  $B$  έχουν όλα τα στοιχεία τους ίδια ή όχι. Συγκεκριμένα, το πρόγραμμά σας πρέπει να:

- Διαβάζει από την πρώτη γραμμή της εισόδου το πλήθος των στοιχείων  $N \leq 50,000$ .
- Διαβάζει από την δεύτερη γραμμή της εισόδου τα  $N$  στοιχεία του πίνακα  $A$  και από την τρίτη γραμμή

του πίνακα  $B$  είναι ακέραιοι αριθμοί που οριζόμενοι). Όλα τα στοιχεία του πίνακα  $A$  και  $B$  είναι διαφορετικά μεταξύ τους.

Αν ναι, το πρόγραμμα τυπώνει τη λέξη ορισμένα με κενό) τη λέξη “no”, το στον άλλο, και το μεγαλύτερο στοιχείο

από 1 sec. Αν δε χρησιμοποιήσετε

```
class queue {
public:
    queue    ();          /* κατασκευαστής: κατασκευάζει μία κενή ουρά */
    bool    empty    ();  /* ελέγχει αν η ουρά είναι κενή */
    void    enqueue  (int x); /* εισάγει ένα στοιχείο στην ουρά */
    int    dequeue   ();  /* αφαιρεί ένα στοιχείο από μια μη κενή ουρά */
    int    peek     ();  /* επιστρέφει (χωρίς να αφαιρεί) το πρώτο στοιχείο μιας μη κενής ουράς */
};
```

Υλοποιήστε αυτόν τον αφηρημένο τύπο δεδομένων χρησιμοποιώντας συνδεδεμένη λίστα.

Στη συνέχεια, χρησιμοποιήστε τον για να γράψετε ένα πρόγραμμα που να διαβάζει μία ακολουθία μη μηδενικών ακέραιων αριθμών (οσοδήποτε πολλών) και να ελέγχει αν αυτή:

- 1) έχει τόσους θετικούς αριθμούς όσους και αρνητικούς, και (συγχρόνως)
- 2) οι αρνητικοί αριθμοί εμφανίζονται με την ίδια σειρά κατ' απόλυτο τιμή που εμφανίζονται και οι θετικοί αριθμοί.

Αν αυτό συμβαίνει, το πρόγραμμα πρέπει να εκτυπώνει το μήνυμα “yes”, διαφορετικά να εκτυπώνει το μήνυμα “no”.

# Μια όχι τόσο καλή ιδέα (για τώρα...)

1

ChatGPT 4o >

Γράψτε ένα πρόγραμμα το οποίο θα βρίσκει και θα εμφανίζει στην οθόνη όλους τους τετραψήφιους ακέραιους που είναι ίσοι με το άθροισμα των ψηφίων τους, υψωμένων στην τέταρτη δύναμη. Π.χ.  $1634 = 1^4 + 6^4 + 3^4 + 4^4$ .

i need a c++ program to solve the above

Here is a C++ program that finds and displays all four-digit numbers that are equal to the sum of their digits raised to the fourth power:

```
cpp Copy
#include <iostream>
#include <cmath>

using namespace std;

bool isValid(int num) {
    int original = num;
    int sum = 0;

    while (num > 0) {
        int digit = num % 10;
        sum += pow(digit, 4);
        num /= 10;
    }

    return sum == original;
}

int main() {
    cout << "Four-digit numbers that satisfy the condition:\n";

    for (int i = 1000; i <= 9999; ++i) {
        if (isValid(i)) {
            cout << i << endl;
        }
    }

    return 0;
}
```

2

3



The screenshot shows a C++ IDE with a project named 'chatKnows1'. The file 'main.cpp' is open, displaying the following code:

```
1 #include <iostream>
2 #include <cmath>
3
4 using namespace std;
5
6 bool isValid(int num) {
7     int original = num;
8     int sum = 0;
9
10    while (num > 0) {
11        int digit = num % 10;
12        sum += pow(x: digit, y: 4);
13        num /= 10;
14    }
15
16    return sum == original;
17 }
18
19 int main() {
20     cout << "Four-digit numbers that satisfy the condition:\n";
21
22     for (int i = 1000; i <= 9999; ++i) {
23         if (isValid(i)) {
24             cout << i << endl;
25         }
26     }
27
28     return 0;
29 }
30
31
```

# Προγραμματισμός - επανάληψη

```
[type] function_name (params)  
{...}
```

**Μονάδες προγράμματος  
(συναρτήσεις)**

```
{ }  
if, if - else, switch  
for(...), while(), do()
```

**Δομές ελέγχου  
εκτέλεσης**

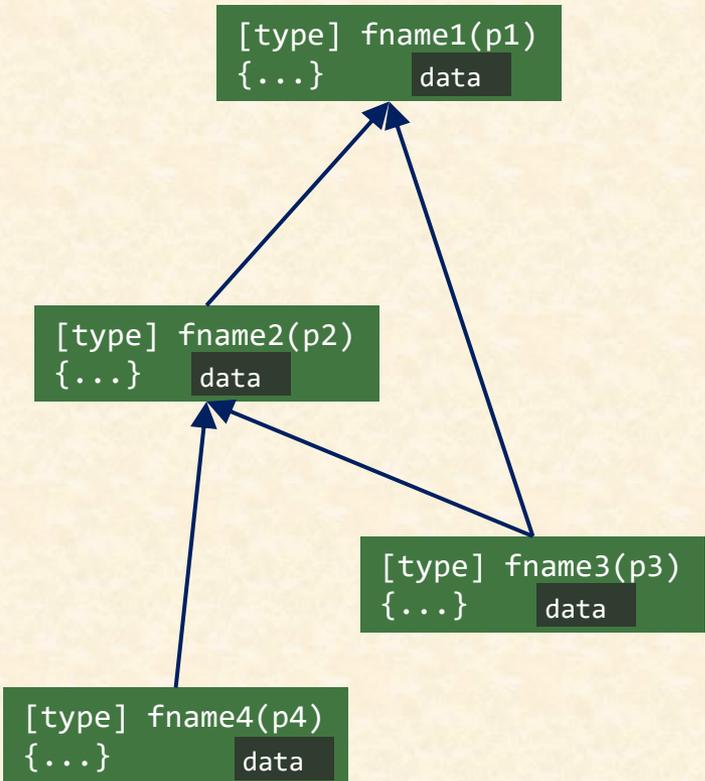
```
int[], float[][], ...  
fstream, ...
```

**Σύνθετα  
δεδομένα**

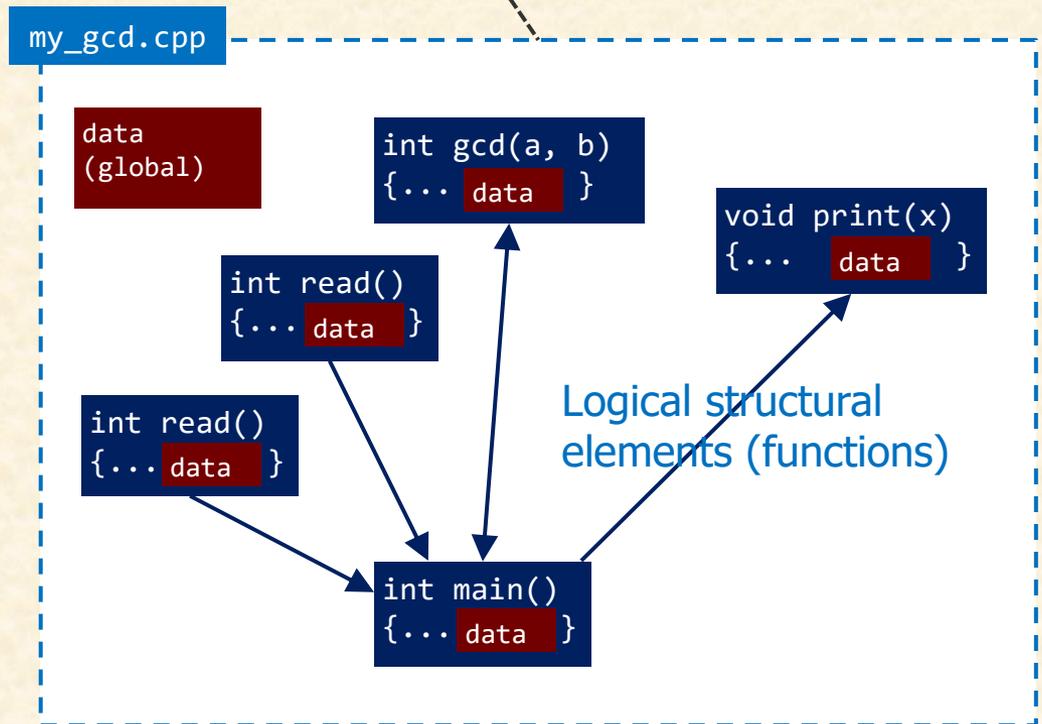
```
int, char, float,  
double, string, ...
```

**Ατομικά  
δεδομένα**

# Προγραμματισμός - επανάληψη



## Source code file (physical container)



# Προγραμματισμός - επανάληψη

my\_gcd.cpp

data  
(global)

```
int gcd(a, b)  
{... data }
```

```
int read()  
{... data }
```

```
void print(x)  
{... data }
```

```
int read()  
{... data }
```

```
int main()  
{... data }
```

```
// function to calc gcd  
int gcd(int x, int y) {  
    while (x > 0 && y > 0)  
        if (x > y)  
            x %= y;  
        else  
            y %= x;  
    return x + y;  
}
```

# Σημεία προσοχής

---

- Ο τρόπος γραφής ενός προγράμματος πρέπει να επιτρέπει να γίνει εύκολα κατανοητή η δομή και η συμπεριφορά του
  - Στοίχιση
  - Σχόλια
  - Ονόματα μεταβλητών
- Η παραγωγή "σωστών αποτελεσμάτων" δεν είναι από μόνη της κριτήριο ποιότητας ενός προγράμματος
- Ο,τι δεν μαθαίνω όταν εκπαιδεύομαι, δύσκολα "θα το κάνω στο μέλλον"

# Ερωτήματα!

---

- Πώς αποφασίζω...
  - Πώς θα οργανώσω τα δεδομένα μου;
  - Πόσες/ποιες συναρτήσεις να κατασκευάσω
  - Ποιες παραμέτρους κάθε συνάρτηση

*Να γράψετε ένα πρόγραμμα που να διαβάζει ένα φυσικό αριθμό  $N > 0$  και να εκτυπώνει το μέγιστο φυσικό αριθμό του οποίου το παραγοντικό δεν υπερβαίνει το  $N$ .*

- «Διαισθητικά» καλές πρακτικές
  - Διαίρει και βασίλευε
  - Μην γράφεις τα πάντα από την αρχή:  
επανα-χρησιμοποίηση

# Ερωτήματα!

```
int main() {
    int N;
    // Εισαγωγή αριθμού από τον χρήστη
    cout << "Δώστε έναν φυσικό αριθμό N (N > 0): ";
    cin >> N;
    if (N <= 0) {
        cout << "Ο αριθμός πρέπει να είναι μεγαλύτερος από το μηδέν!" << endl;
        return 1;
    }
    int k = 1;
    unsigned long long factorial = 1; // Αρχικοποιούμε το παραγοντικό του 1

    while (true) {
        if (factorial > N) { // Αν το παραγοντικό υπερβεί το N, σταματάμε
            k--; // Επιστρέφουμε στο προηγούμενο k που ήταν έγκυρο
            break;
        }
        k++; // Αυξάνουμε το k
        factorial *= k; // Υπολογίζουμε το παραγοντικό του νέου k
    }

    cout << "Ο μέγιστος φυσικός αριθμός του οποίου το παραγοντικό δεν
    υπερβαίνει το " << N << " είναι: " << k << endl;
    return 0;
}
```

ή

```
// Συνάρτηση για τον υπολογισμό του παραγοντικού
unsigned long long factorial(int num) {
    unsigned long long fact = 1;
    for (int i = 2; i <= num; i++) {
        fact *= i;
    }
    return fact;
}

// Συνάρτηση που βρίσκει το μέγιστο k ώστε k! <= N
int findMaxFactorial(int N) {
    int k = 1;
    unsigned long long fact = 1;
    while (fact <= N) {
        k++;
        fact *= k;
        if (fact > N) {
            return k - 1;
        }
    }
    return k - 1;
}

int main() {
    int N;

    // Εισαγωγή αριθμού από τον χρήστη
    cout << "Δώστε έναν φυσικό αριθμό N (N > 0): ";
    cin >> N;

    int maxK = findMaxFactorial(N);
    cout << "Ο μέγιστος φυσικός αριθμός του οποίου το παραγοντικό δεν
    υπερβαίνει το " << N << " είναι: " << maxK << endl;

    return 0;
}
```

# Ερωτήματα!

```
int main() {
    int N;
    // Εισαγωγή αριθμού από τον χρήστη
    cout << "Δώστε έναν φυσικό αριθμό N (N > 0): ";
    cin >> N;
    if (N <= 0) {
        cout << "Ο αριθμός πρέπει να είναι μεγαλύτερος από το μηδέν!" << endl;
        return 1;
    }
    int k = 1;
    unsigned long long factorial = 1; // Αρχικοποιούμε το παραγοντικό του 1

    while (true) {
        if (factorial > N) { // Αν το παραγοντικό υπερβεί το N, σταματάμε
            k--; // Επιστρέφουμε στο προηγούμενο k που ήταν έγκυρο
            break;
        }
        k++; // Αυξάνουμε το k
        factorial *= k; // Υπολογίζουμε το παραγοντικό του νέου k
    }

    cout << "Ο μέγιστος φυσικός αριθμός του οποίου το παραγοντικό δεν
    υπερβαίνει το " << N << " είναι: " << k << endl;
    return 0;
}
```

fact\_n.cpp

```
int main()
{... data }
```

# Ερωτήματα!

```
// Συνάρτηση για τον υπολογισμό του παραγοντικού
unsigned long long factorial(int num) {
    unsigned long long fact = 1;
    for (int i = 2; i <= num; i++) {
        fact *= i; }
    return fact;
}

// Συνάρτηση που βρίσκει το μέγιστο k ώστε k! <= N
int findMaxFactorial(int N) {
    int k = 1;
    unsigned long long fact = 1;
    while (fact <= N) {
        k++;
        fact *= k;
        if (fact > N) {
            return k - 1; }
    }
    return k - 1;
}

int main() {
    int N;

    // Εισαγωγή αριθμού από τον χρήστη
    cout << "Δώστε έναν φυσικό αριθμό N (N > 0): ";
    cin >> N;

    int maxK = findMaxFactorial(N);
    cout << "Ο μέγιστος φυσικός αριθμός του οποίου το παραγοντικό δεν
υπερβαίνει το " << N << " είναι: " << maxK << endl;

    return 0;
}
```

fact\_n\_better.cpp

```
void factorial(x)
{... }
```

```
int findMaxFactorial()
{... }
```

```
int main()
{... }
```

επιστροφή αποτελέσματος  
μετά από κλήση

# Design principles

---

- Αρχή της μοναδικής ευθύνης  
(Single Responsibility Principle – SRP)
  - Code should be focused, narrow, and should do one thing and only one thing well.
  - In object-oriented design, a class should have only and only one responsibility so that it has to change less frequently.
  - Code should be cohesive: it should have one, and only one, reason to change.

# Αρχή της μοναδικής ευθύνης

---

- ...if there is a piece of code that does several things, then that code will need to change often; this is expensive because when a programmer comes to change it, they have to make sure that those changes do not break other things
- ...when a piece of code is broken into several smaller pieces, each one of which does one and only one thing well, then the cost of changes is much lower

# Design principles

---

## □ Open/closed

- A code element (function, class) should be open for extension and closed for modification.
- Code elements should be written in a way that it is ready for adopting/adding new features but "not interested" in any modification.

# Τι κάνει ένα πρόγραμμα «καλό»;

---

- Ορθότητα
  - Ταχύτητα
  - Καλή χρήση πόρων (μνήμη, CPU)
  - Αναγνωσιμότητα
  - Συντηρησιμότητα
  - Επαναχρησιμοποίηση
  - Γρήγορη κατασκευή
- 
- Πολλά από τα παραπάνω συγκρούονται μεταξύ τους (ΠΟΙΑ;)



# Ξεχωριστή μεταγλώττιση

# Ξεχωριστή μεταγλώττιση

---

- Τα προγράμματα συνήθως αποτελούνται από  $> 1$  αρχεία πηγαίου κώδικα και από έτοιμο πηγαίο κώδικα τρίτων ή/και βιβλιοθήκες της γλώσσας
- Κατά τη συγγραφή ενός προγράμματος πρέπει να συμπεριλαμβάνεται στη μεταγλώττιση και σύνδεση μόνο ό,τι απαιτείται (δηλ. όχι τα πάντα)
  - `#include cmath`
  - `#include fstream`
- Η κατάτμηση σε πολλά αρχεία πηγαίου κώδικα επιτρέπει την επαναχρησιμοποίηση (αναγκαία, αλλά όχι ικανή συνθήκη!)

# Ξεχωριστή μεταγλώττιση

(i)

```
int gcd(int x, int y) {                                gcd.cpp
    while (x > 0 && y > 0)
        if (x > y) x %= y; else y %= x;
    return x + y;
}
```

```
#include <iostream>                                    gcd_demo.cpp
using namespace std;

int main() {
    cout << "Give me two numbers: ";
    int a, b;
    cin >> a >> b;
    cout << "GCD (" << a << ", " << b
         << ") = " << gcd(a, b) << endl;
}
```

# Ξεχωριστή μεταγλώττιση

(ii)

- Αρχείο επικεφαλίδας (header file)

```
int gcd(int x, int y); gcd.hpp
```

- Αρχείο υλοποίησης

```
#include "gcd.hpp" gcd.cpp

int gcd(int x, int y) {
    while (x > 0 && y > 0)
        if (x > y) x %= y; else y %= x;
    return x + y;
}
```

# Ξεχωριστή μεταγλώττιση

(iii)

```
int gcd(int x, int y);
```

gcd.hpp

```
#include <iostream>
#include "gcd.hpp"
```

gcd\_demo.cpp

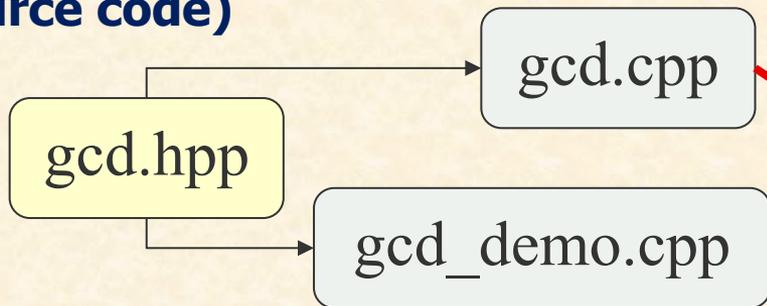
```
using namespace std;
```

```
int main() {
    cout << "Give me two numbers: ";
    int a, b;
    cin >> a >> b;
    cout << "GCD(" << a << ", " << b
         << ") = " << gcd(a, b) << endl;
}
```

# Διαδικασία μεταγλώττισης

(i)

**Πηγαίος κώδικας  
(source code)**



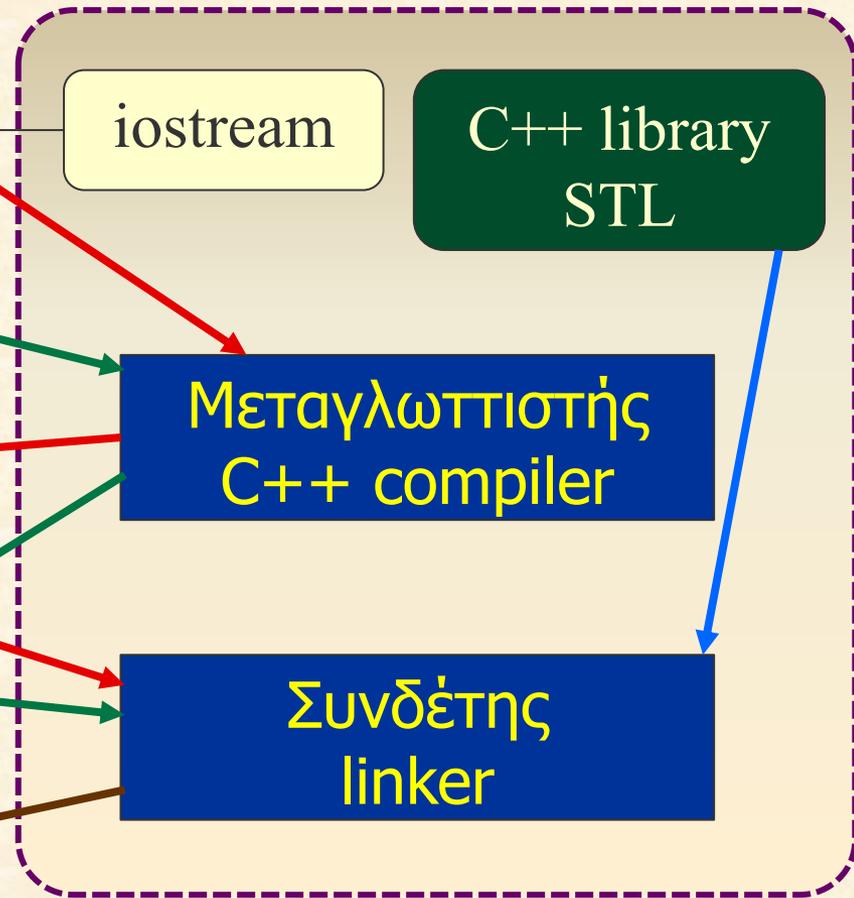
**Object code**

`gcd.o`

`gcd_demo.o`

**Εκτελέσιμο  
(executable)**

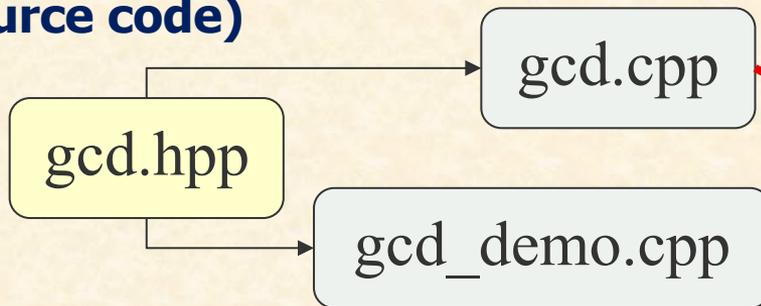
`gcd_demo`



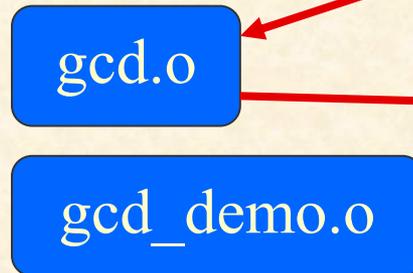
# Διαδικασία μεταγλώττισης

(ii)

Πηγαίος κώδικας  
(source code)



Object code



Εκτελέσιμο  
(executable)



```
$ ls
gcd.hpp
gcd.cpp
gcd_demo.cpp

$ g++ -c gcd.cpp
$ g++ -c gcd_demo.cpp

$ ld -o gcd_demo gcd.o gcd_demo.o
...

$ ./gcd_demo
```

```
$ g++ -o gcd_demo gcd.cpp gcd_demo.cpp
```

# Προγραμματιστικά περιβάλλοντα

```
~/.ssh (-zsh)
+ ~ ssh bxb@novice.softlab.ntua.gr
bxb@novice.softlab.ntua.gr's password:
Linux formal 3.2.0-4-amd64 #1 SMP Debian 3.2.78-1 x86_64

-----
National Technical University of Athens
School of Electrical and Computer Engineering

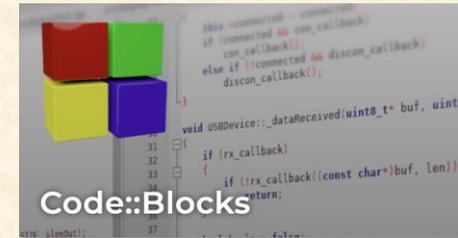
Some graduate, others mature...
-- Yannis Zoulias

Welcome to the server for programming...

-----

No mail.
Last login: Thu Oct 7 16:02:17 2021 from adsl-159.176.58.195.tellas.gr
formal:~>
```

<https://account.jetbrains.com/a/2r3ep14t>





# Πίνακες, revisited

# Πίνακες

(i)

```
int main() {
    int a[10];
    for (int i = 0; i < 10; ++i)
        a[i] = 100*(i+1);
    for (int i = 0; i < 10; ++i)
        cout << a[i] << endl;
}
```

```
const int N = 42;
```

```
int main() { // με σταθερά
    int a[N]; ...
```

```
#define N 42
```

```
int main() { // με μακροεντολή
    int a[N]; ...
```

- Μεγάλοι πίνακες:
  - είτε global (γιατί;;;)
  - είτε με δυναμική παραχώρηση μνήμης:

```
#define N 100000000
int main() {
    int *a = new int[N];
    for (int i = 0; i < N; ++i)
        a[i] = 10*(i+1);
    for (int i = 89; i < 100; ++i)
        cout << a[i] << endl;
    delete [] a;
}
```

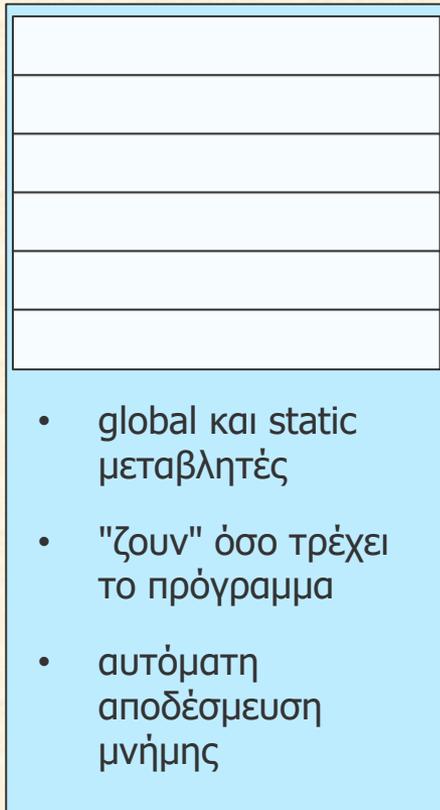
# Δέσμευση μνήμης

---

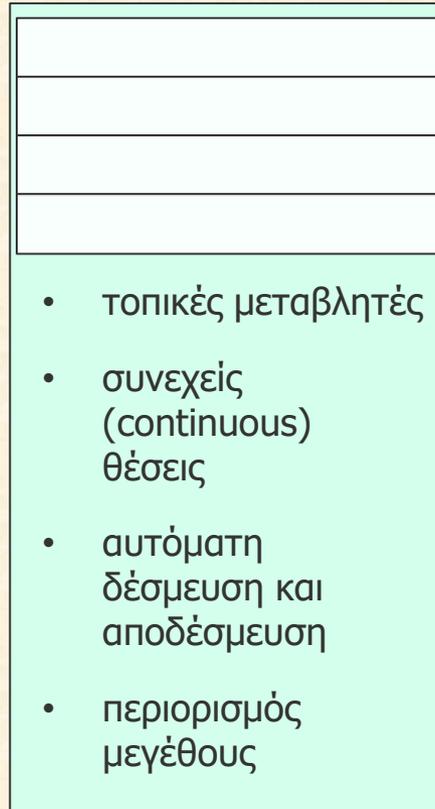
- **Static**
  - global ή static μεταβλητές
- **Stack**
  - μεταβλητές που δηλώνονται μέσα σε συναρτήσεις
- **Heap**
  - μεταβλητές με δυναμική δέσμευση
  - αποδέσμευση από τον προγραμματιστή ή το λειτουργικό, με το πέρας του προγράμματος

# Δέσμευση μνήμης

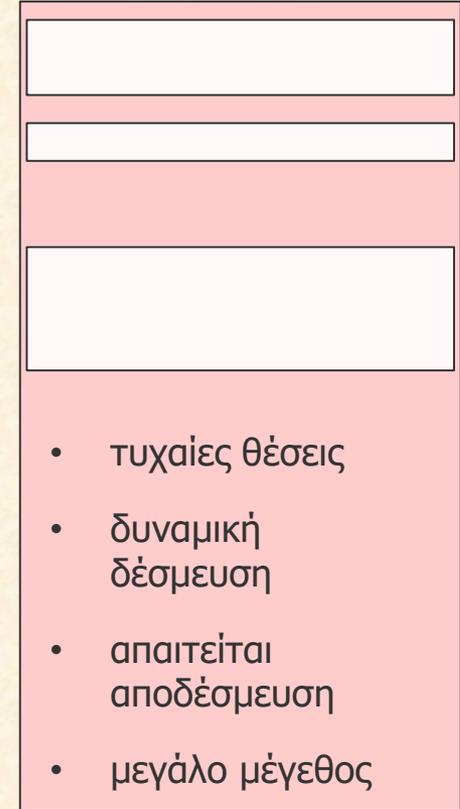
## static



## stack



## heap



# Δυναμική δέσμευση μνήμης

- Αιτούμαστε τη μνήμη και αν υπάρχει παίρνουμε ένα δείκτη σε αυτή (**πού;;;**), διαφορετικά NULL
- Δυναμική δέσμευση σε C: **malloc**

```
int *p;      type casting
...

p = (int *) malloc(sizeof(int));
if (p == NULL) {
    printf("Out of memory! \n");
    exit(1);
}
...
free(p)
```

Ζητούμενο μέγεθος μνήμης

# Δυναμική δέσμευση μνήμης

- Δυναμική δέσμευση σε C++
  - Έλεγχος με ανίχνευση εξαίρεσης (exception) ή με ανίχνευση null, δηλώνοντας nothrow

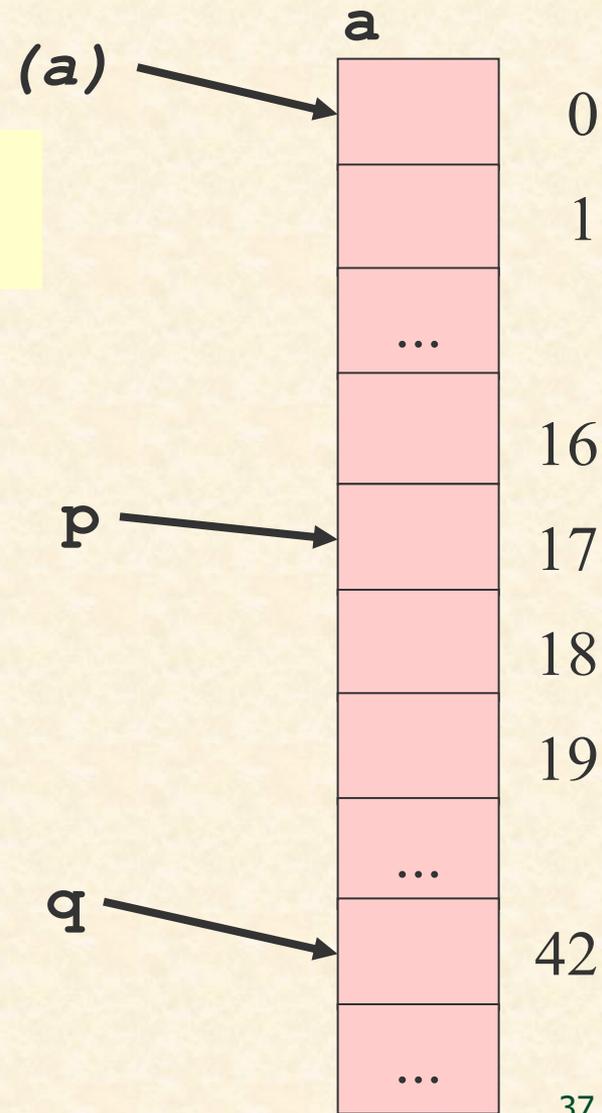
```
int *p = new(nothrow) int;  
if (!p) {  
    cout << "Out of memory!\n";  
}
```

- Συνήθως (;) δεν ελέγχουμε

```
int *a = new int[N];  
for (int i = 0; i < N; ++i) ...
```

# Πίνακες και δείκτες

```
int a[N];  
...  
int *p;           // ισοδύναμα:  
p = &a[17];      int *p = &a[17];  
  
cout << *p << endl;  
cout << *(p+2) << endl;  
cout << p[2] << endl;  
cout << *(p-1) << endl;  
cout << p[-1] << endl;  
  
int *q = &a[42];  
cout << q - p << endl; → 25  
  
p++;  
q -= 4;
```



# Παράδειγμα κώδικα (παίζω και μαθαίνω...)

```
void arraysandpointers() {  
    int a[N] = {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140 };  
  
    cout << "a= " << a << endl;  
    for (int i=0; i<N; i++) { cout << "&a[" << i << "]= " << &a[i] << " a[i]= " << a[i] << endl; }  
  
    int *p;  
    p = &a[1];  
    cout << "p= " << p << endl;  
    cout << "*p= " << *p << endl;  
    cout << "*(p+2)= " << *(p+2) << endl;  
    cout << "p[2]= " << p[2] << endl;  
    cout << "*(p-1)= " << *(p-1) << endl;  
    cout << "p[-1]= " << p[-1] << endl;  
    int *q = &a[4];  
    cout << "q= " << q << endl;  
    cout << "q-p=" << q - p << endl;  
    p++;  
    cout << "p= " << p << " *p= " << *p << endl;  
    q -= 4;  
    cout << "q= " << q << " *q= " << *q << endl;  
}
```

arraysandpointers()



# Πέρασμα παραμέτρων, αναφορές, δείκτες

# Πέρασμα κατ' αναφορά στη C++

(i)

```
void swap(int &x, int &y) {  
    int t = x;  
    x = y;  
    y = t;  
}
```

```
int main() {  
    int a = 42, b = 17;  
    swap(a, b);  
    cout << a << endl;  
    cout << b << endl;  
}
```

# Πέρασμα κατ' αναφορά

(ii)

- Το ίδιο παράδειγμα με δείκτες

```
void swap(int *x, int *y) {  
    int t = *x;  
    *x = *y;  
    *y = t;  
}
```

```
int main() {  
    int a = 42, b = 17;  
    swap(&a, &b);  
    cout << a << endl;  
    cout << b << endl;  
}
```

- Γενικότερος τύπος στη C++,  
όχι μόνο για πέρασμα με αναφορά
- Σαν τους δείκτες αλλά με τις εξής διαφορές:
  - Δε χρειάζονται αστεράκια κατά τη χρήση τους.
  - Όταν δηλώνονται πρέπει οπωσδήποτε να αρχικοποιούνται ώστε να «αναφέρονται» (να δείχνουν) σε κάποια άλλη μεταβλητή.
  - Μετά τη δήλωση και αρχικοποίησή τους, δεν υπάρχει τρόπος να τροποποιηθούν ώστε να αναφέρονται σε άλλη μεταβλητή

# Αναφορές

(ii)

```
int a = 42, b = 17;
```

```
int &c = a; // Οι a και c είναι η ίδια μεταβλητή!
```

```
cout << a << " " << b << " " << c << endl;  
//      42      17      42
```

```
a = 1;  
//      1      17      1
```

```
c = 2;  
//      2      17      2
```

```
c = b;  
//      17     17     17
```

```
c = 3;  
//      3      17     3
```

# Παράδειγμα κώδικα (παίζω και μαθαίνω...)

```
void referencesplay() {
```

```
    int a = 42, b = 17;
```

```
    int &c = a;
```

*// Οι a και c είναι η ίδια μεταβλητή!*

```
    cout << a << " " << b << " " << c << endl; // 42 17 42
```

```
    a = 1;
```

```
    cout << a << " " << b << " " << c << endl; // 1 17 1
```

```
    c = 2;
```

```
    cout << a << " " << b << " " << c << endl; // 2 17 2
```

```
    c = b;
```

```
    cout << a << " " << b << " " << c << endl; // 17 17 17
```

```
    c = 3;
```

```
    cout << a << " " << b << " " << c << endl; // 42 17 3
```

```
}
```

referencesplay()

# Δείκτες και αναφορές σε σταθερές (i)

---

- Μεταβλητές και σταθερές, επανάληψη

```
int a = 42;
```

```
cout << a << endl;
```

```
cin >> a;
```

```
const int b = 42;
```

```
cout << b << endl;
```

```
b = 17;
```

*// απαγορεύεται !!!*

# Δείκτες και αναφορές σε σταθερές (ii)

```
int a = 42;
const int b = 42;

int *p = &a;

cout << *p << endl;
*p = 17;

const int *q = &b;    // δείκτης σε const int

cout << *q << endl;
*q = 17;              // απαγορεύεται !!!

q = &a;

cout << *q << endl;
*q = 17;              // πάλι απαγορεύεται !!!

p = &b;               // απαγορεύεται !!! (γιατί;)
```

# Δείκτες και αναφορές σε σταθερές (iii)

```
int a = 42;
const int b = 42;

int &c = a;

cout << c << endl;
c = 17;

const int &d = b; // αναφορά σε const int

cout << d << endl;
d = 17; // απαγορεύεται !!!

const int &e = a;

cout << e << endl;
e = 17; // πάλι απαγορεύεται !!!

int &f = b; // απαγορεύεται !!! (γιατί;)
```

# Δείκτες και αναφορές σε σταθερές (iv)

## □ Σταθεροί δείκτες

```
int a = 42, z = 0;  
const int b = 42;
```

```
int *p = &a; // δείκτης σε int
```

```
const int *q = &b; // δείκτης σε const int
```

```
int * const r = &a; // σταθερός δείκτης σε int
```

```
r = &z; // απαγορεύεται !!!
```

```
*r = 17; // αυτό επιτρέπεται
```

```
const int * const q = &b; // σταθερός δείκτης  
// σε const int
```

# Παράδειγμα κώδικα (παίζω και μαθαίνω...)

```
void pointersandreferences() {
```

```
    // slide 37
```

```
    int a = 42;  
    const int b = 42;
```

```
    int *p = &a;  
    cout << *p << endl;  
    *p = 17;
```

```
    const int *q = &b; // δείκτης σε const int  
    cout << *q << endl;
```

```
    *q = 17; // απαγορεύεται !!!
```

```
    q = &a;  
    cout << *q << endl;
```

```
    *q = 17; // πάλι απαγορεύεται !!!
```

```
    p = &b; // απαγορεύεται !!! (γιατί;)
```

```
    //-----
```

```
    int &c = a;  
    cout << c << endl;
```

```
    c = 17;  
    const int &d = b; // αναφορά σε const int  
    cout << d << endl;
```

```
    d = 17; // απαγορεύεται !!!  
    const int &e = a;  
    cout << e << endl;
```

```
    e = 17; // πάλι απαγορεύεται !!!  
    int &f = b; // απαγορεύεται !!! (γιατί;)
```

```
    //-----
```

```
    int *p = &a; // δείκτης σε int  
    const int *q = &b; // δείκτης σε const int
```

```
    int z = 0;  
    int * const r = &a; // σταθερός δείκτης σε int  
    r = &z; // απαγορεύεται !!!  
    *r = 17; // αυτό επιτρέπεται  
    const int * const q2 = &b; // σταθερός δείκτης σε  
    const int
```

```
}
```

```
pointersandreferences()
```

# Δείκτες και αναφορές σε σταθερές (v)

```
struct person {
    char firstname[20], lastname[30];
    int phone;
    ...
};

person nickie;

void call(person p) { ... }
...
call(nickie);
```

Η κλήση προκαλεί την αντιγραφή της πραγματικής παραμέτρου **nickie** στην τυπική παράμετρο **p** (call by value!)

# Δείκτες και αναφορές σε σταθερές (vi)

```
struct person {
    char firstname[20], lastname[30];
    int phone;
    ...
};

person nickie;

void call(person &p) { ... }
...
call(nickie);
```

Όχι αντιγραφή (call by reference!) αλλά η συνάρτηση `call` μπορεί να αλλάξει την παράμετρο `nickie`, αν το θέλει

## Δείκτες και αναφορές σε σταθερές (vii)

```
struct person {
    char firstname[20], lastname[30];
    int phone;
    ...
};

person nickie;

void call(const person &p) { ... }
...
call(nickie);
```

Όχι αντιγραφή (call by reference!) **και**  
απαγορεύεται στη συνάρτηση `call` να αλλάξει  
την παράμετρό της

# Και κάποιο συμπέρασμα

---

- Η συνάρτηση μεταβάλλει τις τιμές των παραμέτρων της
  - Πέρασμα ως αναφορά ή με δείκτες
- Η συνάρτηση ΔΕΝ μεταβάλλει τις τιμές των παραμέτρων της
  - **Πέρασμα κατά τιμή**: αντιγραφή της τιμής => κόστος σε χρόνο και μνήμη
  - **Πέρασμα ως const αναφορά**: δεν αντιγράφεται η τιμή => οικονομία σε χρόνο και μνήμη



# Namespaces

# Λίγα λόγια για namespaces

(i)

- Η ζωή χωρίς το **“using namespace std;”**

```
#include <iostream>
```

```
int f(int x) {  
    return x+1;  
}
```

```
int main() {  
    std::cout << f(41) << std::endl;  
}
```

# Λίγα λόγια για namespaces

(ii)

- Πώς να ορίσετε το δικό σας namespace

```
#include <iostream>
```

```
namespace ns {
```

```
int cout(int x) { // Μετονομάζω την 'f' σε 'cout'!  
    return x+1;   // Κακή ιδέα!
```

```
}
```

```
} // namespace ns
```

```
int main() {
```

```
    std::cout << ns::cout(41) << std::endl;
```

```
}
```

# Λίγα λόγια για namespaces

---

## □ QUIZ!

Για ποιο λόγο είναι χρήσιμα τα namespaces

- (α) για έλεγχο της εμβέλειας των μεταβλητών
- (β) για χρήση του ίδιου ονόματος πολλές φορές
- (γ) για επαναχρησιμοποίηση
- (δ) είναι χρήσιμα τα namespaces?!