

# ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

<https://helios.ntua.gr/course/view.php?id=827>

Διδάσκοντες: Στάθης Ζάχος, Νίκος Παπασπύρου,  
Ζωή Παρασκευοπούλου, Δημήτρης Φωτάκης,  
Μάριος Κόνιαρης, Πέτρος Ποτίκας, Δώρα Σούλιου

E-mail προς όλους: [progintro@courses.softlab.ntua.gr](mailto:progintro@courses.softlab.ntua.gr)

## Διαφάνειες παρουσιάσεων

25/9/25

- ✓ Εισαγωγή στην πληροφορική
- ✓ Εισαγωγή στον προγραμματισμό με τη γλώσσα C++
- ✓ Μεθοδολογία αλγορίθμικής επίλυσης προβλημάτων

- Σκοπός του μαθήματος
  - Εισαγωγή στην **πληροφορική** (computer science)
  - Εισαγωγή στον **προγραμματισμό** ηλεκτρονικών υπολογιστών (Η/Υ)
  - Μεθοδολογία **αλγορίθμικής επίλυσης** προβλημάτων

## □ Αλγόριθμος

- Πεπερασμένη ακολουθία ενεργειών που περιγράφει τον τρόπο επίλυσης ενός προβλήματος
- Εφαρμόζεται σε δεδομένα (data)

## □ Πρόγραμμα

- Ακριβής περιγραφή ενός αλγορίθμου σε μια τυπική γλώσσα που ονομάζεται γλώσσα προγραμματισμού

## □ Φυσική γλώσσα

- Χωρίς τόσο αυστηρούς συντακτικούς περιορισμούς
- Μεγάλη πυκνότητα και σημασιολογική ικανότητα

## □ Τυπική γλώσσα

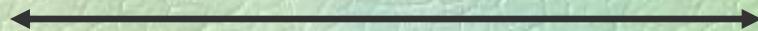
- Αυστηρότατη σύνταξη και σημασιολογία

## □ Γλώσσα προγραμματισμού

- Τυπική γλώσσα στην οποία μπορούν να περιγραφούν υπολογισμοί
- Εκτελέσιμη από έναν ηλεκτρονικό υπολογιστή

## □ Πληροφορική

Ηλεκτρονικοί  
υπολογιστές  
(engineering)



Μαθηματικά

Σχεδίαση και  
κατασκευή

Θεωρία και  
αναλυτική μέθοδος

## □ Κεντρική έννοια: υπολογισμός (computation)

- Πληροφορική: μαθηματικοποίηση της μεθοδολογίας των μηχανικών
  - Απαιτήσεις – Πρόβλημα
  - Προδιαγραφές
  - Σχεδίαση
  - Υλοποίηση
  - Εμπειρικός έλεγχος – Θεωρητική επαλήθευση
  - Βελτιστοποίηση
  - Πολυπλοκότητα (κόστος πόρων-αγαθών)
  - Τεκμηρίωση
  - Συντήρηση

Έννοιες που υπήρχαν για τους μηχανικούς, στην πληροφορική τυποποιήθηκαν, πήραν μαθηματική μορφή, άρα μπορεί κανείς να επιχειρηματολογήσει με αυτές τις έννοιες χρησιμοποιώντας αποδείξεις.

## □ Δευτεροβάθμια εκπαίδευση

**Σκοπός:** να μάθεις να σκέφτεσαι

- Η Ευκλείδεια Γεωμετρία (με τη βασική διδακτική της αξία) απουσιάζει από το πρόγραμμα σπουδών εδώ και χρόνια.
- Αποτέλεσμα: όπως είδαμε και στις πανελλήνιες εξετάσεις δίνεται έμφαση στην αποστήθιση ανουσίων θεωρημάτων και γνώσεων διαφορικού και απειροστικού λογισμού. Η ικανότητα μαθηματικής επίλυσης απλών αλλά πρωτότυπων προβλημάτων δεν παίζει ρόλο.
- Απουσία γνώσεων συνδυαστικής (μέτρηση περιπτώσεων, τρίγωνο Pascal).
- Εφαρμογή των αποστηθισμένων κανόνων;
- Άλγεβρα: αν ρωτήσω έναν τελειόφοιτο Λυκείου πόσο κάνει  $107 \times 93$  θα δυσκολευτεί πολύ να απαντήσει, ενώ φυσικά γνωρίζει ότι  $(\alpha+\beta)(\alpha-\beta) = \alpha^2 - \beta^2$

- Οι μαθητές αγνοούν την έννοια του “αποδοτικού αλγόριθμου”
  - π.χ. μαθαίνουν ένα μη-αποδοτικό αλγόριθμο για την εύρεση του Μ.Κ.Δ. ενώ ο αλγόριθμος του Ευκλείδη απουσιάζει από την ύλη
- Πρόταση
  - Εισαγωγή της Θεωρητικής Πληροφορικής στη δευτεροβάθμια εκπαίδευση για όλους τους μαθητές
  - Μεθοδολογία επίλυσης προβλημάτων με σχεδίαση και υλοποίηση αλγορίθμων

## □ Τριτοβάθμια εκπαίδευση

- Η τεχνολογία αλλάζει αέναα και γρήγορα – τα θεμέλια μένουν
- Αυτά τα θεμέλια πρέπει να είναι η ραχοκοκαλιά στην τριτοβάθμια εκπαίδευση: έμφαση στην αλγορίθμική σκέψη σε αντιδιαστολή με τις τεχνολογικές δεξιότητες (computer literacy)
- Computer science, computing science, informatics
- **Dijkstra**: η Επιστήμη των Υπολογιστών έχει τόση σχέση με τους υπολογιστές όση και η Αστρονομία με τα τηλεσκόπια
- **Primality**: σημαντικό επίτευγμα σε μία χώρα χωρίς υποδομές

## □ Να μην ξεχνάμε ότι

- Το να κάνεις λάθη είναι ανθρώπινο.
- Για να τα κάνεις θάλασσα χρειάζεσαι υπολογιστή!

## □ Κατασκευή υπολογιστικών μηχανών

- **Αρχαιότητα:** υπολογιστικές μηχανές, μηχανισμός των Αντικυθήρων, κ.λπ.
- 17ος αιώνας, **Pascal** και **Leibniz**, μηχανικές υπολογιστικές αριθμομηχανές  
⇒ στοιχειώδεις αριθμητικές πράξεις
- 1830–1840, **Babbage**, “αναλυτική μηχανή”  
⇒ λογάριθμοι, τριγωνομετρικές συναρτήσεις
- 1880–1890, **Hollerith**, μηχανή με διάτρητες κάρτες για την αυτοματοποίηση των εκλογών

## □ Κατασκευή υπολογιστών

- 1920–1930, Bush, ηλεκτρική (αναλογική) υπολογιστική μηχανή  $\Rightarrow$  διαφορικές εξισώσεις
- ~1940, Zuse, ηλεκτρονική (ψηφιακή) υπολογιστική μηχανή $\Rightarrow$  πρόγραμμα και δεδομένα, χωριστά
- 1945–1950, μοντέλο von Neumann $\Rightarrow$  πρόγραμμα και δεδομένα, από κοινού
- 1950–σήμερα, ραγδαία ανάπτυξη της τεχνολογίας των ηλεκτρονικών υπολογιστών

## □ Κατασκευή υπολογιστών

- ▶ 1952– main frames IBM 650, 7000, 360
- ▶ 1965– mini computers DEC PDP-8
- ▶ 1977– personal computers Apple II
- ▶ 1981 IBM PC
- ▶ 1983,  
1984 Apple: Lisa, Macintosh
- ▶ 1985– internet
- ▶ 1990– world wide web
- ▶ 2000– PDA, smartphones, cloud, κ.λπ.

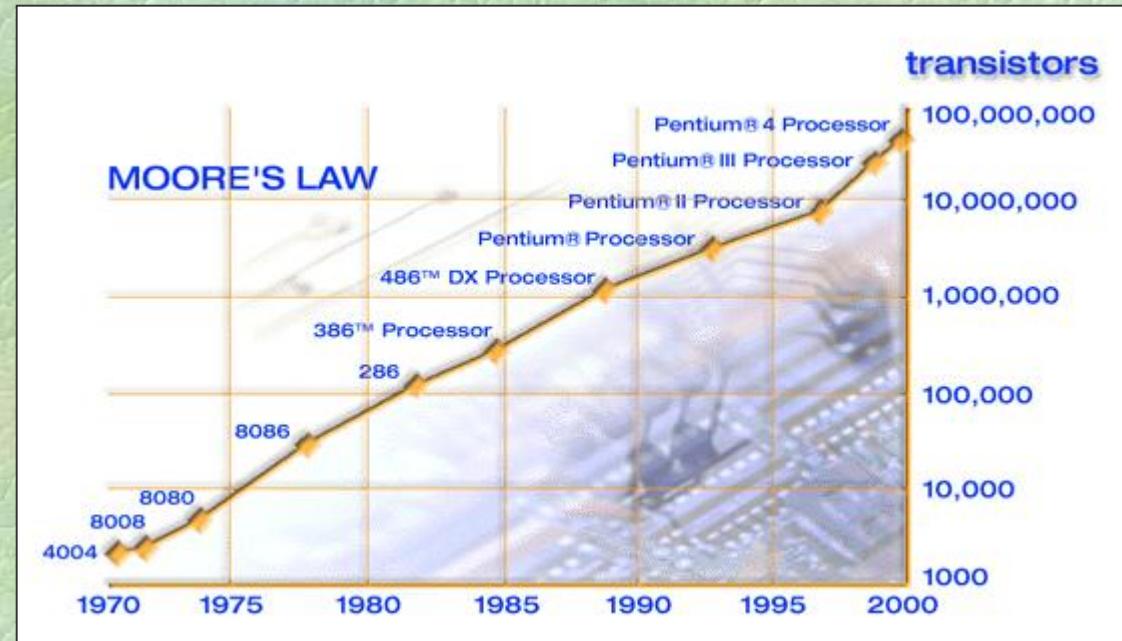
## □ Μηχανικοί υπολογιστών

- Tom Watson, IBM, 1945

*Ο κόσμος χρειάζεται περίπου 5 υπολογιστές*

- Gordon Moore, Intel, 1965

*Η πυκνότητα του hardware στα ολοκληρωμένα κυκλώματα διπλασιάζεται κάθε 18 μήνες*



<http://www.intel.com/research/silicon/mooreslaw.htm>

## □ Θεμέλια της πληροφορικής

- Μαθηματική λογική
- Αριστοτέλης: συλλογισμοί

$$\frac{A \quad A \rightarrow B}{B} \quad (modus\ ponens)$$

- Ευκλείδης: αξιωματική θεωρία
- Αρχές 20ου αιώνα, Hilbert  
⇒ αξίωμα, θεώρημα, τυπική απόδειξη

## □ Πρόγραμμα του Leibniz:

θεμελίωση των μαθηματικών

- γλώσσα για όλα τα μαθηματικά
- θεωρία
- συνεπής (consistent) και πλήρης (complete)

$$A \wedge \neg A \quad \text{αντίφαση}$$

## □ Γλώσσα (Boole, De Morgan, Frege, Russel)

- προτασιακός λογισμός  $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$
- κατηγορηματικός λογισμός  $\forall, \exists$

## □ Θεωρία

- Συνολοθεωρία, Cantor, Frege ∈
- Παράδοξο του Russel

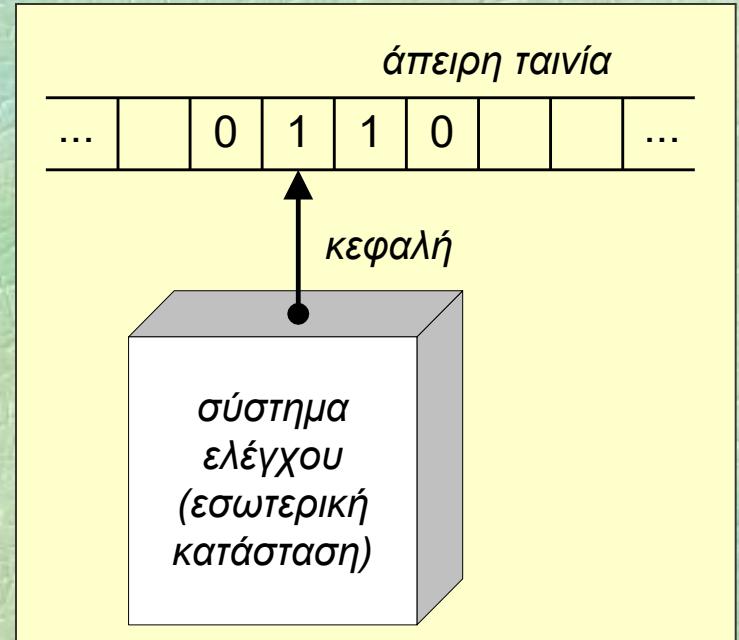
$$A = \{ x \mid x \notin x \}$$

$$\begin{aligned} A \in A &\rightarrow A \notin A \\ A \notin A &\rightarrow A \in A \end{aligned}$$

- Άλλες θεωρίες συνόλων (ZF, κ.λπ.)
- Άλλες θεωρίες για τη θεμελίωση των μαθηματικών (θεωρία συναρτήσεων, κατηγοριών, κ.λπ.)
- 1920–1930, προσπάθειες για απόδειξη συνέπειας

## □ Συνέπεια και πληρότητα

- 1931, Gödel, θεώρημα μη πληρότητας  
⇒ δεν είναι δυνατόν να κατασκευαστεί συνεπής και πλήρης θεωρία της αριθμητικής
- 1936, Turing,  
⇒ μη αποκρίσιμες (undecidable) προτάσεις  
⇒ μηχανή Turing, υπολογισιμότητα



## □ Μη πληρότητα (incompleteness)

- David Hilbert, 1862-1943
- Kurt Gödel, 1906-1978 (ασιτία)
- Δοξιάδης
  - Incompleteness: a play and a theorem
  - Ο θείος Πέτρος και η εικασία του Goldbach
- Παπαδημητρίου
  - Το χαμόγελο του Turing
- Hoffstader
  - Gödel, Escher, and Bach

## □ Κλάδοι της πληροφορικής

- Αλγόριθμοι και δομές δεδομένων
- Γλώσσες προγραμματισμού
- Αρχιτεκτονική υπολογιστών και δικτύων
- Αριθμητικοί και συμβολικοί υπολογισμοί
- Λειτουργικά συστήματα
- Μεθοδολογία – τεχνολογία λογισμικού
- Βάσεις δεδομένων και διαχείριση πληροφοριών
- Τεχνητή νοημοσύνη και ρομποτική
- Επικοινωνία ανθρώπου – υπολογιστή

## □ Υπολογιστής

- επεξεργαστής
- μνήμη
- συσκευές εισόδου/εξόδου

## □ Ιδιότητες

- αυτόματο χωρίς εξυπνάδα
- μεγάλη ταχύτητα
- ακρίβεια στις πράξεις

# Γλώσσες προγραμματισμού (i)

## □ Γλώσσα μηχανής

0110110 11011011

διεύθυνση εντολή

## □ Συμβολική γλώσσα (assembly)

label: add ax, bx

διεύθυνση πράξη δεδομένα

## □ Γλώσσες χαμηλού και υψηλού επιπέδου

## □ Υλοποίηση γλωσσών προγραμματισμού

□ μεταγλωττιστής (compiler)

□ διερμηνέας (interpreter)

## □ Κυριότερες γλώσσες, ιστορικά

1950

1960      FORTRAN, LISP, COBOL, Algol,  
              BASIC, PL/I

1970      **Pascal, C,**

              Smalltalk, Prolog, ML, Logo

1980      **C++, Modula-2, Ada, Perl**

1990      Java, Python, Ruby, Haskell, PHP

2000      C#, ...

# Γλώσσες προγραμματισμού (iii)

## □ Pascal

- Niklaus Wirth (1971)
- Γλώσσα γενικού σκοπού (general purpose)
- Συστηματικός και δομημένος προγραμματισμός

## □ C

- Dennis Ritchie (1972)
- Γενικού σκοπού αλλά χαμηλότερου επιπέδου
- Προγραμματισμός συστημάτων

## □ C++

- Bjarne Stroustrup (1983)
- Γενικού σκοπού, αντικειμενοστρεφής



```
#include "pzhelp"  
  
PROGRAM {  
    WRITELN("hello world");  
}
```

#1

```
#include "pzhelp"  
  
PROGRAM {  
    WRITELN("hello", "world");  
}
```

```
#include "pzhelp"  
  
PROGRAM {  
    WRITE("hello "); WRITELN("world");  
}
```

```
#include "pzhelp"  
  
PROGRAM {  
    WRITE("hello world"); WRITELN();  
}
```

# Δυο λόγια για το `#include "pzhelp"`

```
#include <iostream>
using namespace std;

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

Απλή C++

```
#include "pzhelp"
```

```
PROGRAM {
```

```
    WRITELN("hello world");
```

```
}
```

C++ & **pzhe1p**

```
#include "pzhelp"

void hello() {
    WRITELN("hello world");
}

PROGRAM {
    hello(); hello();
    hello(); hello();
}
```

#2

```
#include "pzhelp"

void hello() {
    WRITELN("hello world");
}

PROGRAM {
    for (int i = 0; i < 20; i++)
        hello();
}
```

#3

```
#include "pzhelp"  
const int n = 20;  
int i;  
  
void num_hello() {  
    WRITELN(i, "hello world");  
}  
  
PROGRAM {  
    for (i = 1; i <= n; i++)  
        num_hello();  
}
```

#4

```
#include "pzhelp"

void hello() {
    WRITELN("hello world");
}

PROGRAM {
    WRITELN("Give number of greetings",
            "then press <enter>:");
    int n = READ_INT();
    for (int i = 0; i < n; i++) hello();
}
```

#5

#6

```
#include "pzhelp"

void hello() {
    WRITELN("hello world");
}

PROGRAM {
    WRITE("Give number of greetings",
          "then press <enter>: ");
    int n = READ_INT();
    if (n < 0)
        WRITELN("# is negative");
    else
        for (int i = 0; i < n; i++)
            hello();
}
```

# Δομή του προγράμματος

```
#include "pzhelp"
```

```
PROGRAM
```

επικεφαλίδα

```
{
```

```
REAL r, a;
```

δηλώσεις

```
WRITE ("Give the radius: ");
```

```
r = READ_REAL();
```

```
a = 3.1415926 * r * r;
```

```
WRITELN ("The area is:", a);
```

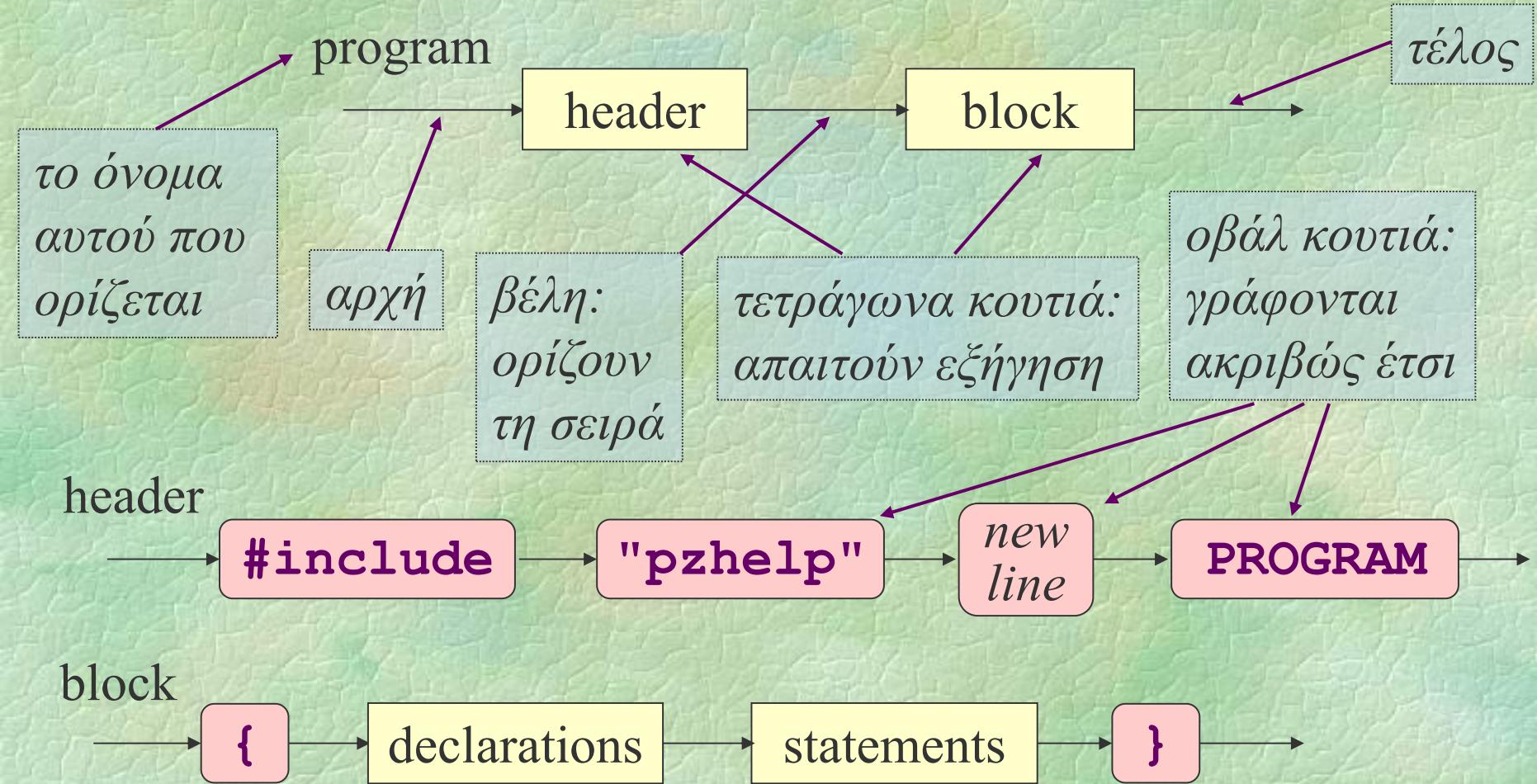
εντολές

```
}
```

σώμα = block

# Συντακτικά διαγράμματα

Περιγράφουν τη **σύνταξη** τμημάτων του προγράμματος



# μεταβλητή

?

!

- **μεταβλητή**: ένα «κουτί» της μνήμης του υπολογιστή όπου μπορεί να αποθηκευτεί μια πληροφορία (ένα δεδομένο)
- στο τμήμα δηλώσεων ορίζουμε **όλες** τις μεταβλητές που χρησιμοποιεί το πρόγραμμα
- για κάθε μεταβλητή ορίζουμε το **όνομά** της και τον **τύπο** της, δηλαδή το πεδίο των τιμών που μπορεί να πάρει η μεταβλητή

**int i;**

# Δηλώσεις μεταβλητών (ii)

## □ Απλοί τύποι μεταβλητών

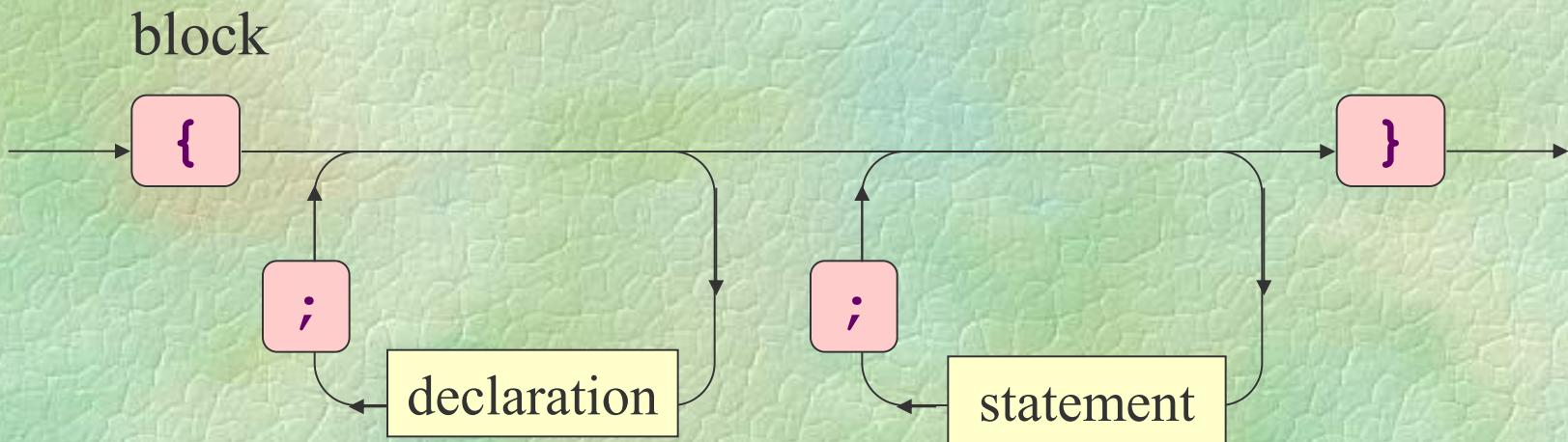
□ <b>int</b>	<i>ακέραιοι αριθμοί</i>	0, 1, -3
□ <b>REAL</b>	<i>πραγματικοί αριθμοί</i>	3.14
□ <b>char</b>	<i>χαρακτήρες</i>	'a'
□ <b>bool</b>	<i>λογικές τιμές</i>	<b>true, false</b>

## □ Δήλωση περισσότερων μεταβλητών

```
int i, j, k;  
REAL x, y;           char ch;  
bool changed;
```

# Συντακτικά διαγράμματα, ξανά

- Αναλυτικότερο συντακτικό διάγραμμα για το σώμα (block)



# Σχόλια

- Μίας ή περισσότερων γραμμών

```
REAL x, y; /* οι συντεταγμένες  
του κέντρου */  
REAL r; // η ακτίνα
```

- Γράφονται (σχεδόν) παντού και αγνοούνται από τον μεταγλωττιστή
- Βοηθούν στην κατανόηση του κώδικα

# Τι σημαίνει ορθό πρόγραμμα (i)

## □ Συντακτική ορθότητα

- το πρόγραμμα πρέπει να υπακούει στους συντακτικούς κανόνες της γλώσσας προγραμματισμού

## □ Συντακτικά σφάλματα στη C++

- εμφανίζονται όταν δεν ικανοποιούνται τα συντακτικά διαγράμματα
- παράδειγμα:  
`{ PROGRAM } )WRITELN; ("hello world"`

# Τι σημαίνει ορθό πρόγραμμα (ii)

## □ Νοηματική ορθότητα

- το πρόγραμμα πρέπει να υπακούει τους νοηματικούς κανόνες της γλώσσας προγραμματισμού

## □ Νοηματικά σφάλματα στη C++

- εσφαλμένη χρήση τελεστών  
`n = "a" * 3;`
- χρήση μεταβλητών χωρίς δήλωση  
`int n, i;`  
`n = i + j;`

# Τι σημαίνει ορθό πρόγραμμα (iii)

## □ Σημασιολογική ορθότητα

- όταν το πρόγραμμα εκτελείται, πρέπει να κάνει ακριβώς αυτό που θέλουμε να κάνει

## □ Σημασιολογικά σφάλματα στη C++

- προέρχονται από την κακή σχεδίαση ή την κακή υλοποίηση του προγράμματος
- αυτά τα σφάλματα ονομάζονται συνήθως bugs και η διαδικασία εξάλειψής τους debugging

`x1 = (-b + sqrt(b*b-4*a*c)) / (2*a);`

`sqrt`

*διαίρεση με  
το μηδέν*

# Τι σημαίνει ορθό πρόγραμμα (iv)

- Ο μεταγλωττιστής μπορεί να εντοπίσει σε ένα πρόγραμμα την ύπαρξη
  - συντακτικών σφαλμάτων
  - νοηματικών σφαλμάτων
- Τυπώνει κατάλληλα μηνύματα σφάλματος
- Ο προγραμματιστής είναι υπεύθυνος για
  - τη διόρθωση των παραπάνω
  - τον εντοπισμό και τη διόρθωση σημασιολογικών σφαλμάτων

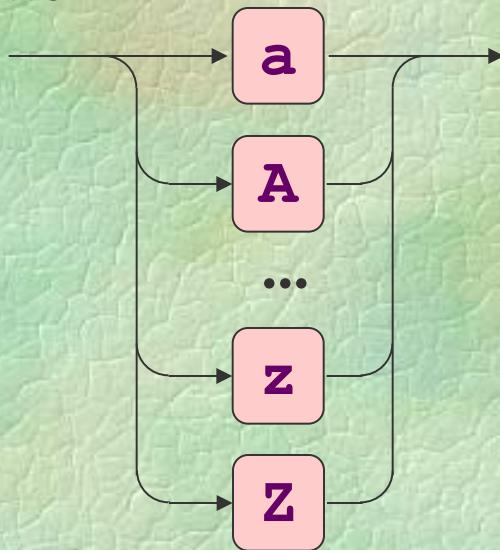
# Συντακτικά διαγράμματα, ξανά

identifier

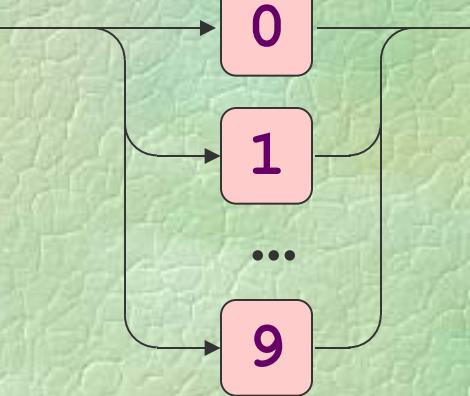


πεζά και κεφαλαία γράμματα  
είναι διαφορετικά!

letter



digit



# Ανάθεση τιμής σε μεταβλητή

## □ Παραδείγματα αναθέσεων

```
n = 2;
```

```
pi = 3.14159;
```

```
done = true;
```

```
ch = 'b';
```

```
counter = counter + 1;
```

```
x1 = (-b + sqrt(b*b-4*a*c)) / (2*a);
```

# Έξοδος στην οθόνη (i)

## □ Έξοδος στην οθόνη

```
WRITELN ("Hello world!");
```

```
WRITELN ("Hello", "world!");
```

## □ Έξοδος χωρίς αλλαγή γραμμής

```
WRITE ("Hel");
```

```
WRITELN ("lo", "world!");
```

## □ Έξοδος στην οθόνη

```
x = 6;
```

```
WRITE ("x");
```

```
WRITE (" = ");
```

```
WRITE (x);
```

```
WRITELN();
```

```
WRITELN ("3*x-1 =", 3*x-1);
```

```
WRITELN ("x*(x+1) =", x*(x+1));
```

**x = 6**

**3\*x-1 = 17**

**x\*(x+1) = 42**

## □ Έξοδος στην οθόνη

```
WRITE (4) ;
```

42

```
WRITELN (2) ;
```

4 2

```
WRITELN (4, 2) ;
```

666

```
WRITE (6) ;
```

6 66

```
WRITE (6) ;
```

```
WRITELN (6) ;
```

```
WRITE (6, 6) ;
```

```
WRITELN (6) ;
```

# Είσοδος από το πληκτρολόγιο (i)

## □ Είσοδος από το πληκτρολόγιο

```
n = READ_INT();  
r = READ_REAL();  
c = getchar();
```

## □ Είσοδος από το πληκτρολόγιο και διάβασμα μέχρι το τέλος της γραμμής

```
SKIP_LINE();
```

# Είσοδος από το πληκτρολόγιο (ii)

```
PROGRAM {
    REAL r, a;
    WRITE ("Give the radius: ");
    r = READ_REAL ();
    a = 3.1415926 * r * r;
    WRITELN ("The area is:", a);
}
```

# Είσοδος από το πληκτρολόγιο (iii)

```
PROGRAM {  
    WRITE ("Give the radius: ");  
    REAL r = READ_REAL ();  
    REAL a = 3.1415926 * r * r;  
    WRITELN ("The area is:", a);  
}
```

Ταυτόχρονη δήλωση μεταβλητών και  
“αρχικοποίηση” (initialization)  
= ανάθεση για πρώτη φορά

# Είσοδος από το πληκτρολόγιο (iv)

```
PROGRAM {
    int first = READ_INT();
    char operation = getchar();
    int second = READ_INT();

    int result;

    switch (operation) {
        case '+': result = first + second; break;
        case '-': result = first - second; break;
        case '*': result = first * second; break;
        case '/': result = first / second; break;
    }
    WRITELN("The result is:", result);
}
```

# Είσοδος από το πληκτρολόγιο (v)

## □ Αποθηκευτικός χώρος (buffer)

- παρεμβάλλεται μεταξύ του πληκτρολογίου και του προγράμματος
- εκεί αποθηκεύονται **προσωρινά** τα δεδομένα που πληκτρολογεί ο χρήστης μέχρι να διαβαστούν από το πρόγραμμα
- η εισαγωγή στο buffer γίνεται με το πάτημα του πλήκτρου **enter**
- αρχικά ο buffer είναι κενός

# Είσοδος από το πληκτρολόγιο (vi)

```
PROGRAM {
    WRITE("First: ");
    int first = READ_INT();
    WRITE("Second: ");
    int second = READ_INT();
    WRITELN("Result:", first + second);
}
```

# Είσοδος από το πληκτρολόγιο (vii)

## □ Πρώτη εκτέλεση παραδείγματος

First: 3 ↵

Second: 6 ↵

Result: 9

## □ Δεύτερη εκτέλεση παραδείγματος

First: 3 6 ↵

Second: Result: 9

# Είσοδος από το πληκτρολόγιο (viii)

```
PROGRAM {
    WRITE("First: ");
    int first = READ_INT();
    SKIP_LINE();
    WRITE("Second: ");
    int second = READ_INT();
    WRITELN("Result:", first + second);
}
```

# Είσοδος από το πληκτρολόγιο (ix)

## □ Πρώτη εκτέλεση παραδείγματος

First: 3 ↵

Second: 6 ↵

Result: 9

## □ Δεύτερη εκτέλεση παραδείγματος

First: 3 6 ↵

Second: 7 ↵

Result: 10

## □ Απλές παραστάσεις

- σταθερές και μεταβλητές

## □ Απλές πράξεις

- |                                |                   |
|--------------------------------|-------------------|
| □ πρόσθεση, αφαίρεση           | + , -             |
| □ πολλαπλασιασμός              | *                 |
| □ διαίρεση πραγματικών αριθμών | /                 |
| □ πηλίκο ακέραιας διαίρεσης    | / <i>προσοχή!</i> |
| □ υπόλοιπο ακέραιας διαίρεσης  | % <i>modulo</i>   |
| □ πρόσημα                      | + , -             |

## □ Παραδείγματα

- $1 + 1 \rightarrow 2$  τύπου **int**
- $1.0 + 2.0 \rightarrow 3.0$  τύπου **REAL**
- $1 + 3.0 \rightarrow 4.0$  τύπου **REAL**
- $5 / 2 \rightarrow 2$  ακέραιο πηλίκο
- $5 \% 2 \rightarrow 1$  ακέραιο υπόλοιπο
- $5.0 / 2 \rightarrow 2.5$  διαίρεση **REAL**
- $5.0 \% 2 \rightarrow \cancel{}$  απαγορεύεται!

## □ Πραγματική διαίρεση μεταξύ ακεραίων;

► **int**  $x=42, y=17;$  **WRITE** ( **1.0 \* x / y** );

# Αριθμητικές παραστάσεις (iii)

## □ Προτεραιότητα τελεστών

□ π.χ.  $5+3*x-y \equiv 5+(3*x)-y$

## □ Προσεταιριστικότητα τελεστών

□ π.χ.  $x-y+1 \equiv (x-y)+1$

## □ Σειρά εκτέλεσης των πράξεων

- καθορίζεται **εν μέρει** από την προτεραιότητα  
και την προσεταιριστικότητα των τελεστών
- γενικά όμως εξαρτάται από την υλοποίηση
- π.χ.  $(x+1) * (y-1)$

# □ Συγκρίσεις

- ισότητα, ανισότητα ==, !=
  - μεγαλύτερο, μικρότερο >, <
  - μεγαλύτερο ή ίσο, μικρότερο ή ίσο >=, <=

## □ Λογικές πράξεις

- σύζευξη (καὶ)      &&      *and*
  - διάζευξη (ή)      | |      *or*
  - ἀρνηση (όχι)      !      *not*

## □ Πίνακες αλήθειας λογικών πράξεων

<b>p</b>	<b>q</b>	<b>p &amp;&amp; q</b>
<b>false</b>	<b>false</b>	<b>false</b>
<b>false</b>	<b>true</b>	<b>false</b>
<b>true</b>	<b>false</b>	<b>false</b>
<b>true</b>	<b>true</b>	<b>true</b>

σύζευξη

<b>p</b>	<b>q</b>	<b>p    q</b>
<b>false</b>	<b>false</b>	<b>false</b>
<b>false</b>	<b>true</b>	<b>true</b>
<b>true</b>	<b>false</b>	<b>true</b>
<b>true</b>	<b>true</b>	<b>true</b>

διάζευξη

<b>p</b>	<b>! p</b>
<b>false</b>	<b>true</b>
<b>true</b>	<b>false</b>

άρνηση

# Προτεραιότητα τελεστών

(i)

Τελεστής	Σημασία	Προσεταιριστικότητα
+ - !	πρόσημα, λογική άρνηση	—
* / %	πολλαπλασιασμός, διαίρεση	αριστερά
+ -	πρόσθεση, αφαίρεση	αριστερά
< <= >= >	σύγκριση	αριστερά
== !=	ισότητα	αριστερά
&&	λογική σύζευξη	αριστερά
	λογική διάζευξη	αριστερά

επάνω: μεγάλη προτεραιότητα

κάτω: μικρή προτεραιότητα

# Προτεραιότητα τελεστών (ii)

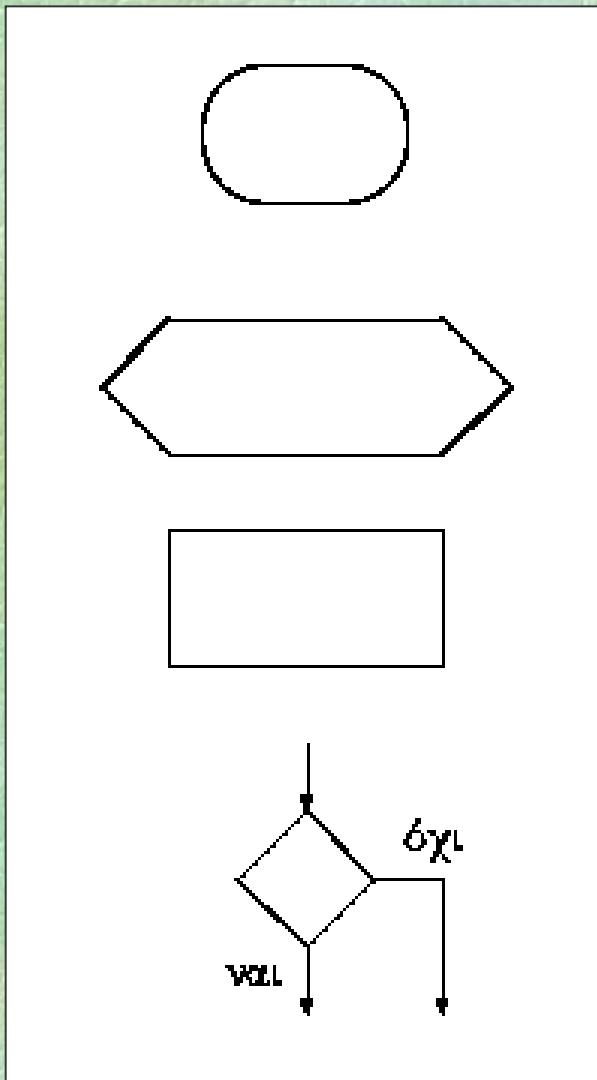
## □ Προτεραιότητα τελεστών

- $\pi.\chi. \text{ !p } \&\& \text{ q } || \text{ r }$   
 $\equiv ((\text{ !p }) \&\& \text{ q }) \ || \text{ r }$
- $\pi.\chi. \text{ x>3 } \&\& \text{ ! y+z==5 }$  λάθος!  
 $\equiv (\text{ x>3 }) \&\& (((\text{ !y }) + z) == 5)$
- $\pi.\chi. \text{ x>3 } \&\& \text{ ! (y+z==5) }$  σωστό  
 $\equiv (\text{ x>3 }) \&\& (\text{ ! } ((\text{ y+z }) == 5))$
- Όταν δεν είμαστε σίγουροι, δε βλάπτει να χρησιμοποιούμε επιπλέον παρενθέσεις!

# Δομές ελέγχου

- Τροποποιούν τη σειρά εκτέλεσης των εντολών του προγράμματος
- Οι εντολές φυσιολογικά εκτελούνται κατά σειρά από την αρχή μέχρι το τέλος
- Με τις δομές ελέγχου επιτυγχάνεται:
  - ομαδοποίηση εντολών
  - εκτέλεση εντολών υπό συνθήκη
  - επανάληψη εντολών

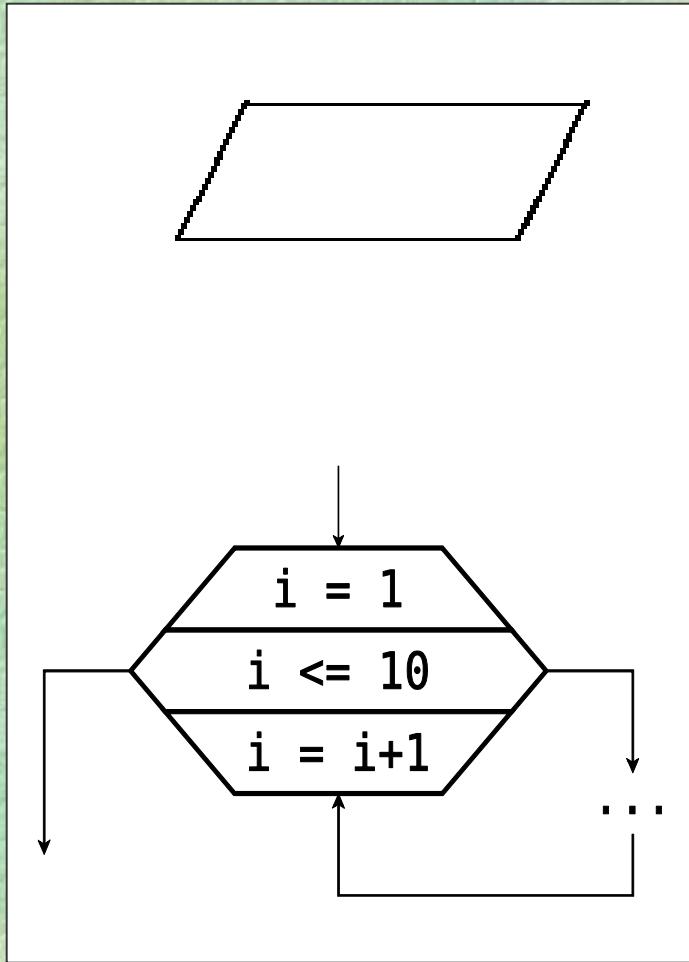
# Λογικά διαγράμματα ροής (i)



- Αρχή και τέλος
- Ολόκληρες λειτουργίες ή διαδικασίες
- Απλές εντολές
- Έλεγχος συνθήκης

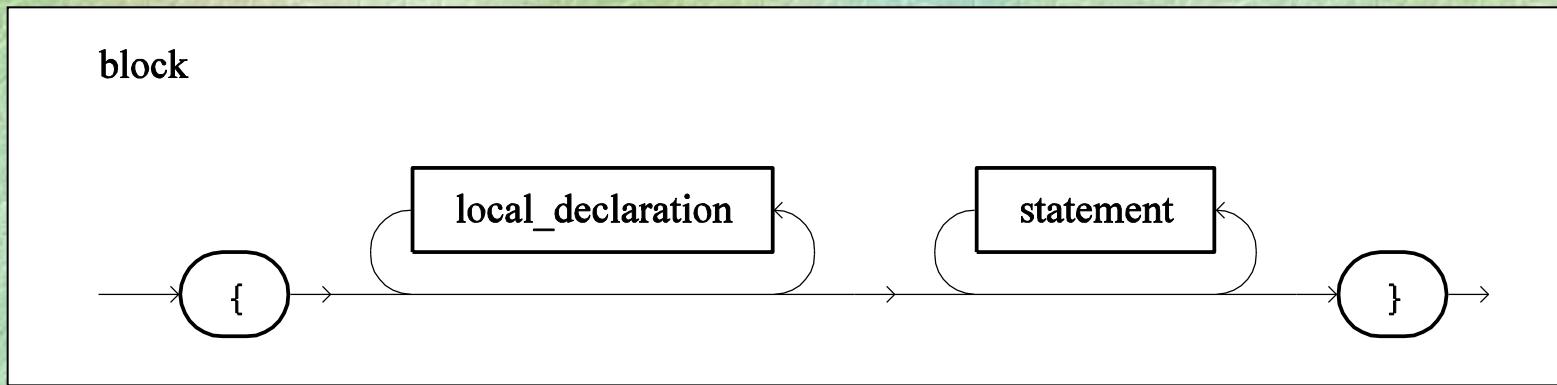
# Λογικά διαγράμματα ροής

(ii)



- Λειτουργία εισόδου/εξόδου
- Επανάληψη (βρόχος)

- Ομαδοποίηση πολλών εντολών σε μία
- Χρήσιμη σε συνδυασμό με άλλες δομές
- Συντακτικό διάγραμμα



## □ Παραδείγματα

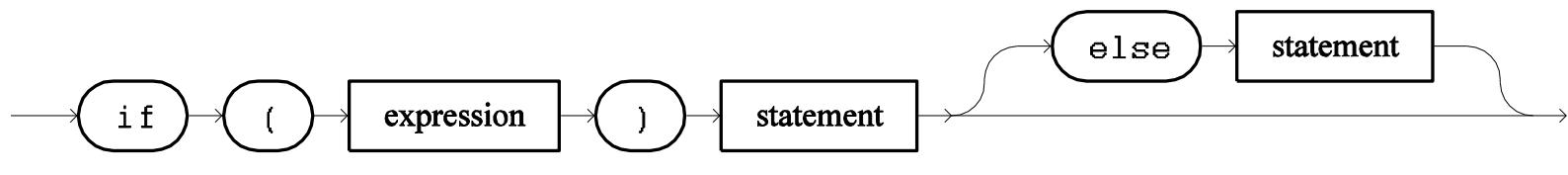
```
{  
    int x=2, y=3, z=3;  
    WRITELN(x, y, z);  
}  
  
{  
    a=2; b=3;  
    {  
        c=3;  
        WRITE(a, b, c);  
    }  
    WRITELN();  
}
```

# Εντολή if

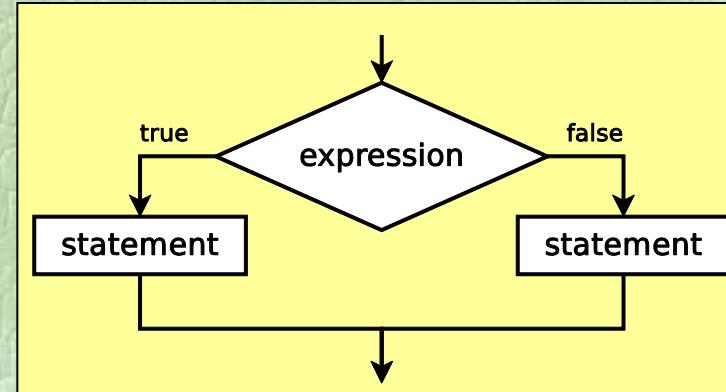
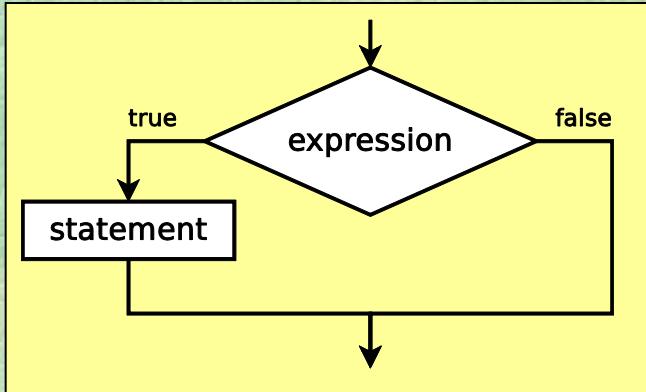
(i)

- Εκτέλεση εντολών υπό συνθήκη
- Συντακτικό διάγραμμα

if\_stmt



- Λογικό διάγραμμα



# Εντολή if

---

(ii)

## □ Παραδείγματα

```
if (amount >= x) amount = amount - x;  
  
if (amount >= 1000000)  
    WRITELN("Found a millionaire!");  
  
if (year > 1900 && year <= 2000)  
    WRITE("20ός αιώνας");  
  
if (x*x + y*y == z*z) {  
    WRITELN("Pythagorian:", x, y, z);  
    s = (z-x)*(z-y)/2;  
    WRITELN("Perfect square:", s);  
}
```

# Εντολή if

## (iii)

### □ Παραδείγματα

```
if (year % 4 == 0 &&
    year % 100 != 0 ||
    year % 400 == 0 &&
    year % 4000 != 0)
WRITELN("Το έτος", year,
        "είναι δίσεκτο!");
```

# Εντολή if (iv)

## □ Παραδείγματα (συνέχεια)

```
if (x % 2 == 0)  WRITELN("άρτιος");
else
    WRITELN("περιττός");

if (x > y)          WRITELN("κέρδισα");
else if (x < y)
    WRITELN("κέρδισες");
else                  WRITELN("ισοπαλία");
```

το παρακάτω είναι ισοδύναμο αλλά χειρότερο:

```
if (x > y)          WRITELN("κέρδισα");
if (x < y)          WRITELN("κέρδισες");
if (x == y)          WRITELN("ισοπαλία");
```

# Εντολή if (v)

- Ένα **else** αντιστοιχεί στο πλησιέστερο προηγούμενο **if** που δεν έχει ήδη αντιστοιχιστεί σε άλλο **else**
- Παράδειγμα

```
if (x > 0)
    if (y > 0)
        WRITELN("πρώτο τεταρτημόριο");
    else if (y < 0)
        WRITELN("τέταρτο τεταρτημόριο");
else
    WRITELN("άξονας των x");
```

# Εντολή switch (i)

- Εκτέλεση υπό συνθήκη για πολλές διαφορετικές περιπτώσεις
- Προσφέρεται π.χ. αντί του:

```
if (month == 1)
    WRITELN("Ιανουάριος");
else if (month == 2)
    WRITELN("Φεβρουάριος");
else if ...
    ...
else if (month == 12)
    WRITELN("Δεκέμβριος");
else
    WRITELN("άκυρος μήνας");
```

## □ Παραδείγματα

```
switch (month) {  
    case 1: WRITELN ("Ιανουάριος") ;  
    break;  
    case 2: WRITELN ("Φεβρουάριος") ;  
    break;  
    ...  
    case 12: WRITELN ("Δεκέμβριος") ;  
    break;  
    default: WRITELN ("άκυρος μήνας") ;  
    break;  
}
```

## □ Περισσότερα παραδείγματα

```
switch (month) {  
    case 1: case 3: case 5: case 7:  
    case 8: case 10: case 12:  
        WRITELN("31 days"); break;  
    case 4: case 6: case 9: case 11:  
        WRITELN("30 days"); break;  
    case 2:  
        WRITELN("28 or 29 days"); break;  
}
```

- Οι μέρες μέχρι την πρωτοχρονιά

```
r = 0;
```

```
switch (month) {  
    case 1: r = r + 31; // συνεχίζει...  
    case 2: r = r + 28; // συνεχίζει...  
    case 3: r = r + 31; // συνεχίζει...  
    case 4: r = r + 30; // συνεχίζει...  
    case 5: r = r + 31; // συνεχίζει...  
    ...  
    case 11: r = r + 30; // συνεχίζει...  
    case 12: r = r + 31; // συνεχίζει...  
}  
r = r - day + 1;  
WRITELN("Μένουν", r, "μέρες!");
```

Χωρίς break

# Εντολή for

(i)

□ Μαθαίνω να μετράω

```
PROGRAM { // counting
    int i;
    WRITELN("Look:");
    for (i = 1; i <= 10; i++)
        WRITELN(i);
}
```

Look:  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10

1

Ξεκίνησε τη μεταβλητή **i**  
από την τιμή 1

2

Όσο η τιμή της **i**  
δεν ξεπερνά το 10

3

Εκτέλεσε  
το σώμα

4

Αύξησε κατά 1  
την τιμή της **i**

5

Επανάλαβε!

- Μαθαίνω να μετράω

```
PROGRAM { // counting
    WRITELN("Look:");
    for (int i = 1; i <= 10; i++)
        WRITELN(i);
}
```

- Η μεταβλητή **i** ορίζεται και μπορεί να χρησιμοποιηθεί μόνο μέσα στο βρόχο

### □ Δυνάμεις του δύο

```
PROGRAM { // powers of two
    int p = 1;
    for (int i = 0; i <= 10; i++) {
        WRITELN(2, "^", i, "=", p);
        p = p * 2;
    }
}
```

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$

$$\text{Αναλλοίωτη: } p = 2^i$$

# Εντολή for (iv)

## □ Παραγοντικό

$$n! = 1 \times 2 \times 3 \times \dots \times n$$

$$0! = 1$$

```
PROGRAM { // factorial
    WRITE("Give n: ");
    int n = READ_INT();
    int p = 1;
    for (int i = 2; i <= n; i++)
        p = p * i;
    WRITELN(n, " ! =", p);
}
```

*Anaλλοιωτη:  $p = i!$*

Give n: 1	↓
1 ! = 1	
Give n: 4	↓
4 ! = 24	
Give n: 7	↓
7 ! = 5040	
Give n: 12	↓
12 ! = 479001600	
Give n: 17	↓
17 ! = -288522240	

*Υπερχείλιση!*

## □ Βλέπω αστεράκια

```
for (i = 1; i <= 5; i++) {  
    for (j = 1; j <= 10; j++)  
        WRITE("*");  
    WRITELN();  
}
```

A rectangular grid of 50 black asterisks (\*). It consists of 5 rows, each containing 10 asterisks. This represents the output of the first code snippet.

```
*****  
*****  
*****  
*****  
*****
```

```
for (i = 1; i <= 5; i++) {  
    for (j = 1; j <= 2*i; j++)  
        WRITE("*");  
    WRITELN();  
}
```

A triangular grid of 50 black asterisks (\*). The pattern forms a right-angled triangle pointing upwards, with 1 asterisk in the first row, 2 in the second, 3 in the third, 4 in the fourth, and 5 in the fifth. This represents the output of the second code snippet.

```
**  
***  
****  
*****  
*****
```

## □ Ντόμινο

- οι αριθμοί πηγαίνουν μέχρι το  $n = 6$
- συνολικά 28 κομμάτια
- τι γίνεται για άλλες τιμές του  $n$ ;



# Εντολή for

(vii)

```
PROGRAM { // domino2
    WRITE("Give n: ");
    int n = READ_INT();
    int count = 0;
    for (int i = 0; i <= n; i++)
        for (int j = i; j <= n;
            j++) {
            WRITELN(i, j);
            count = count + 1;
        }
    WRITELN("Total", count,
            "pieces.");
}
```

```
Give n: 3↵
0 0
0 1
0 2
0 3
1 1
1 2
1 3
2 2
2 3
3 3
Total 10 pieces.
```

# Εντολή for

(viii)

- Ακριβώς  $i+1$  κομμάτια έχουν τον αριθμό  $i$  ως μεγαλύτερο!

```
PROGRAM { // dominol
    WRITE("Give n: ");
    int n = READ_INT();
    int count = 0;
    for (int i = 0; i <= n;
         i++) {
        WRITELN(i+1, "with largest", i);
        count = count + i + 1;
    }
    WRITELN("Total", count, "pieces.");
}
```

```
Give n: 6↙
1 with largest 0
2 with largest 1
3 with largest 2
4 with largest 3
5 with largest 4
6 with largest 5
7 with largest 6
Total 28 pieces.
```

- Λίγα μαθηματικά δε βλάπτουν...

$$count = \sum_{i=0}^n (i+1) = \sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$$

```
PROGRAM { // domino0
    WRITE("Give n: ");
    int n = READ_INT();
    int count = (n+2)*(n+1)/2;
    WRITELN("Total", count, "pieces.");
}
```

```
Give n: 6↙
Total 28 pieces.

Give n: 17↙
Total 171 pieces.

Give n: 42↙
Total 946 pieces.
```

# Εντολή for (x)

- Υπολογίζουμε το ίδιο με 3 διαφορετικούς τρόπους

$$count = \sum_{i=0}^n \sum_{j=i}^n 1 = \sum_{i=0}^n (i+1) = \frac{(n+1)(n+2)}{2}$$

**domino2    domino1    domino0**

- Πόσες αριθμητικές πράξεις κάνουν;
  - **domino2**:  $(n+1)(n+2)/2$  προσθέσεις  $O(n^2)$
  - **domino1**:  $2(n+1)$  προσθέσεις  $O(n)$
  - **domino0**: 2 προσθέσεις, 1 πολλαπλασιασμός  $O(1)$
- Καλύτερο (γρηγορότερο) πρόγραμμα:  
αντό με τη μικρότερη δυνατή πολυπλοκότητα!
- Πόσο σκέφτομαι εγώ / Πόσο «σκέφτεται» ο Η/Υ !

# Εντολή for (xi)

## □ Παραλλαγές: αύξηση και μείωση

- **for** (*i* = **lower**; *i* <= **upper**; *i*++)  
αν **lower** ≤ **upper**, θα γίνουν **upper** – **lower** + 1 επαναλήψεις, αλλιώς καμία
- **for** (*i* = **upper**; *i* >= **lower**; *i*--)  
αν **lower** ≤ **upper**, θα γίνουν **upper** – **lower** + 1 επαναλήψεις, αλλιώς καμία

- Παραλλαγές: αύξηση και μείωση με βήμα
  - **for** (*i* = 1; *i* <= 20; *i* += 3)  
*i* παίρνει τις τιμές: 1, 4, 7, 10, 13, 16, 19
  - **for** (*i* = 100; *i* >= 50; *i* -= 5)  
*i* παίρνει τις τιμές: 100, 95, 90, 85, ..., 60, 55, 50

- Ειδικές περιπτώσεις: μία φορά

```
for (i = 10; i <= 10; i++)
```

```
for (i = 10; i >= 10; i--)
```

- Ειδικές περιπτώσεις: καμία φορά

```
for (i = 12; i <= 10; i++)
```

```
for (i = 10; i >= 12; i--)
```

## □ Κακή ιδέα #1:

- Να αλλάζουμε τη μεταβλητή ελέγχου  
(π.χ. με ανάθεση) μέσα στο σώμα του βρόχου
  - ▶ **for** (i = 1; i <= 10; i++) {
  - ▶   **if** (i > 5) **i = 20;**
  - ▶   **WRITELN(i);**
  - ▶ }

## □ Δεν απαγορεύεται, κάνει όμως δύσκολη την κατανόηση των βρόχων!

**for**: συγκεκριμένο, εκ των προτέρων γνωστό πλήθος επαναλήψεων!

## □ Κακή ιδέα #2:

- Να αλλάζουμε το (άνω ή κάτω) όριο (π.χ. με ανάθεση) μέσα στο σώμα του βρόχου

► n = 10;

```
► for (i = 1; i <= n; i++) {
```

▶ n--;

n++;

WRITELN (i) ;

{

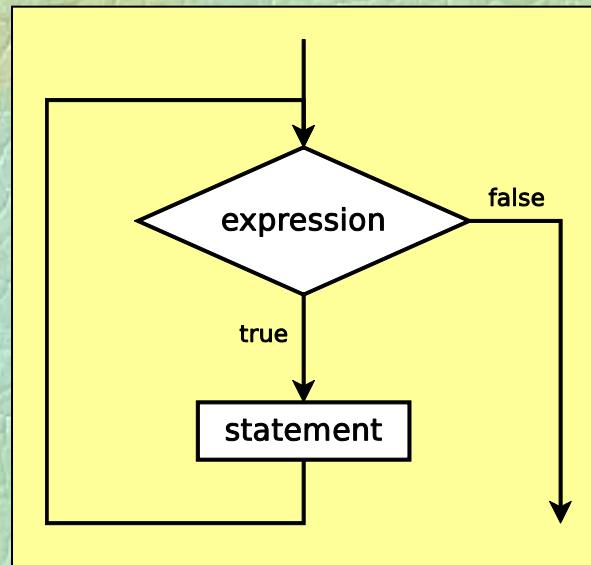
□ Ούτε αυτό απαγορεύεται, κάνει όμως δύσκολη την κατανόηση των βούγουν

**for**: συγκεκριμένο, εκ των προτέρων γνωστό πλήθος επαναλήψεων!

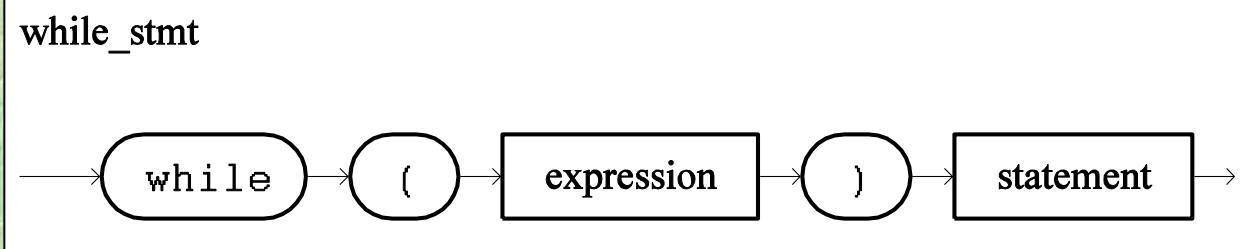
# Εντολή while

(i)

- Βρόχος όσο ικανοποιείται μια συνθήκη
- Λογικό διάγραμμα



- Συντακτικό διάγραμμα



- Ο αριθμός επαναλήψεων γενικά δεν είναι γνωστός εκ των προτέρων
- Αν η συνθήκη είναι αρχικά ψευδής, ο βρόχος τερματίζεται χωρίς να εκτελεστεί το σώμα
- Η ροή ελέγχου μπορεί να μεταβληθεί με τις εντολές **break** και **continue**

# Εντολή while (iii)

□ Δυνάμεις του δύο, ξανά

```
PROGRAM { // more powers of two
    int p = 1, i = 0;
    while (p <= 10000000) {
        WRITELN(2, "^", i,
                 "=", p);
        p = p * 2;
        i = i + 1;
    }
}
```

Αναλλοίωτη:  $p = 2^i$

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
...
$2^{22} = 4194304$
$2^{23} = 8388608$

## □ Άπειρος βρόχος

```
PROGRAM { // line punishment
    while (true)
        WRITELN("I must not tell lies");
}
```

```
I must not tell lies
I must not tell lies
I must not tell lies
...
```

Break

Διακόπτουμε ένα πρόγραμμα με **Ctrl+C** ή **Ctrl+Break**

- Άπειρος βρόχος, λιγότερο προφανής

```
PROGRAM {
    // another_infinite_loop
    int x = 17;
    while (x > 0)
        x = (x + 42) % 2024;
}
```

*Αναλλοίωτη: το x είναι θετικός και περιττός ακέραιος*

x
17
59
101
143
185
...
1991
9
51
93
...

# Εντολή while

(vi)

## □ Πρώτοι αριθμοί

```
PROGRAM { // primes
    WRITELN(2);
    for (int p = 3; p < 1000;
         p += 2) {
        int t = 3;
        while (p % t != 0)
            t = t + 2;
        if (p == t) WRITELN(p);
    }
}
```

<u>Output</u>	p	t
2	3	3
3	5	3
5	5	5
7	7	3
11	5	5
...	7	7
997	9	3
	11	3
		5
		7
...	...	...
997	997	997
999		3

Αναλλοίωτη του **while**: το  $p$  δε διαιρείται με κανέναν αριθμό  $\geq 2$  και  $\leq t$

- Μέγιστος κοινός διαιρέτης των  $a$  και  $b$ ,  
ένας απλός αλγόριθμος

```
z = min(a, b);  
while (a % z != 0 || b % z != 0)  
    z = z - 1;  
writeln(z);
```

*Αναλλοίωτη: δεν υπάρχει αριθμός  $w > z$   
που να διαιρεί και τον  $a$  και τον  $b$*

*Πολυπλοκότητα:  $O(\min(a, b))$*

- Μέγιστος κοινός διαιρέτης των  $a$  και  $b$ ,  
αλγόριθμος με αφαιρέσεις
  - Ιδέα 1: αν  $a > b$  τότε  $\text{gcd}(a, b) = \text{gcd}(a-b, b)$
  - ```
while (a > 0 && b > 0)
    if (a > b) a = a - b; else b = b - a;
writeln(a+b);
```
- Στη χειρότερη περίπτωση, η πολυπλοκότητα  
είναι τώρα  $O(\max(a, b))$
- Σε «τυπικές» περίπτωσεις όμως, αυτός ο  
αλγόριθμος είναι καλύτερος του προηγούμενου

- Μέγιστος κοινός διαιρέτης των  $a$  και  $b$ ,  
αλγόριθμος του Ευκλείδη
  - Ιδέα 2: αν  $a > b$  τότε  $\text{gcd}(a, b) = \text{gcd}(a \bmod b, b)$
  - while** ( $a > 0$  **&&**  $b > 0$ )  
    **if** ( $a > b$ )  $a = a \% b$ ; **else**  $b = b \% a$ ;  
**WRITELN** ( $a+b$ );
  - $\text{gcd}(54, 16) = \text{gcd}(6, 16) = \text{gcd}(6, 4) =$   
 $\text{gcd}(2, 4) = \text{gcd}(2, 0) = 2$
  - $\text{gcd}(282, 18) = \text{gcd}(12, 18) = \text{gcd} (12, 6) =$   
 $\text{gcd}(0, 6) = 6$
  - Πολυπλοκότητα:  $O(\log(a+b))$

# Εντολή while (x)

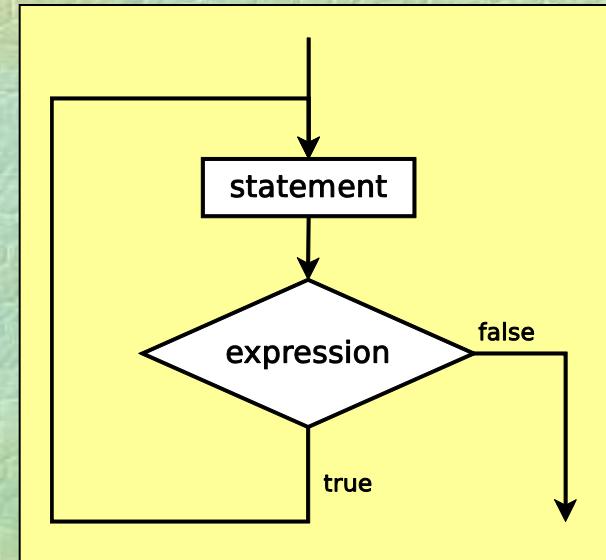
```
PROGRAM { // gcd
    WRITE("Give a: ");
    int a = READ_INT();
    WRITE("Give b: ");
    int b = READ_INT();
    WRITE("gcd(", a, ", ", b, ") =");
    a = abs(a); b = abs(b);

    while (a > 0 && b > 0)
        if (a > b) a = a % b; else b = b % a;
    WRITELN(a+b);
}
```

# Εντολή do ... while

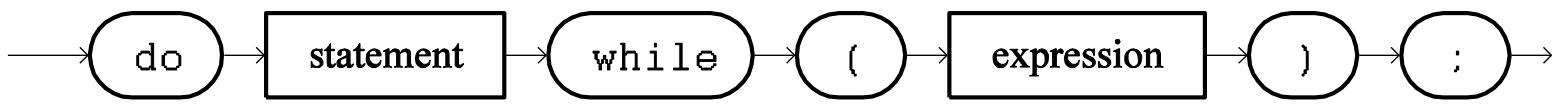
(i)

- Βρόχος με τη συνθήκη να αποτιμάται στο τέλος κάθε επανάληψης
- Λογικό διάγραμμα



- Συντακτικό διάγραμμα

do\_while\_stmt



- Ο έλεγχος της συνθήκης γίνεται στο τέλος κάθε επανάληψης (και όχι στην αρχή)
- Το σώμα του βρόχου εκτελείται τουλάχιστον μία φορά
- Ο αριθμός επαναλήψεων γενικά δεν είναι γνωστός εκ των προτέρων
- Η ροή ελέγχου μπορεί να μεταβληθεί με τις εντολές **break** και **continue**

## □ Αριθμοί Fibonacci

$$F_0 = 0, \quad F_1 = 1$$

$$F_{n+2} = F_n + F_{n+1}, \quad \forall n \in \mathbf{N}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,  
233, 377, 610, 987, 1597, 2584, 4181, ...

□ **Πρόβλημα:** ποιος είναι ο μεγαλύτερος αριθμός Fibonacci που δεν υπερβαίνει το  $n$ ;

$$F_k \leq n \text{ και } F_{k+1} > n$$

□ **NB:** Η ακολουθία Fibonacci είναι αύξουσα

# Εντολή do ... while

## (iv)

```
PROGRAM { // fibonacci
    WRITE("Give n: ");
    int n = READ_INT();
    if (n <= 1) WRITELN(n);
    else {
        int prev = 0, curr = 1;
        do {
            int next = curr + prev;
            prev = curr;
            curr = next;
        } while (curr <= n);
        WRITELN(prev);
    }
}
```

```
Give n: 20↙
13
Give n: 100↙
89
Give n: 987↙
987
```

*Αναλλοίωτη;*

# Εντολή do ... while

(v)

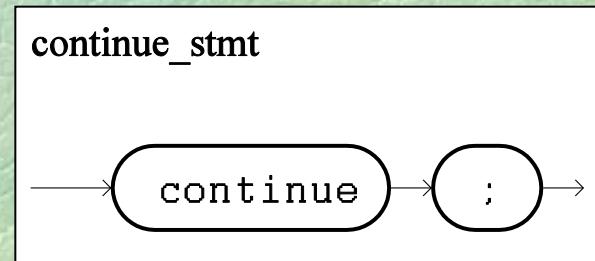
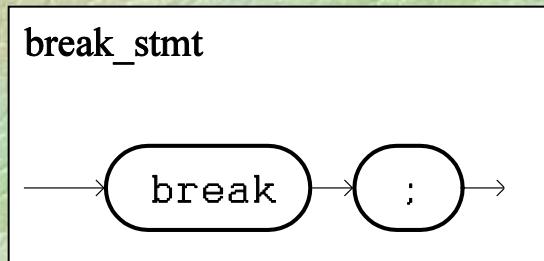
```
PROGRAM { // bigsum
    do {
        int sum = 0;
        char symbol;
        do {
            int number = READ_INT();
            sum = sum + number;
            do symbol = getchar();
            while (symbol != '+' &&
                   symbol != '=');
        } while (symbol == '+');
        WRITELN(sum);
    } while (true);
}
```

8+  
9=  
17  
6+  
3+  
12+  
21=  
42

Break

# Εντολές break και continue (i)

- Η **break** προκαλεί τον άμεσο (πρόωρο) τερματισμό ενός βρόχου
- Η **continue** προχωράει αμέσως στην επόμενη επανάληψη ενός βρόχου



# Εντολές break και continue (ii)

## □ Η ατυχής εικασία...

Ένας φίλος μας μαθηματικός ισχυρίζεται ότι για κάθε πρώτο αριθμό  $p$  ισχύει:

$$(17p) \text{ mod } 4217 \neq 42$$

- Θα προσπαθήσουμε να βρούμε αντιπαράδειγμα!
- Δηλαδή έναν πρώτο αριθμό  $p$  τέτοιον ώστε
$$(17p) \text{ mod } 4217 = 42$$

# Εντολές break και continue (iii)

- Θα τροποποιήσουμε το πρόγραμμα υπολογισμού των πρώτων αριθμών

```
PROGRAM { // primes
    WRITELN(2);
    for (int p = 3; p < 1000; p += 2) {
        int t = 3;
        while (p % t != 0) t = t + 2;
        if (p == t) WRITELN(p);
    }
}
```

# Εντολές break και continue

(iv)

```
PROGRAM { // prime_conj
    for (int p = 3; p < 1000000;
          p += 2) {
        int t = 3;
        while (p % t != 0) t = t+2;
        if (p != t) continue;
        if ((17 * p) % 4217 == 42) {
            WRITELN("Counterexample:", p);
            break;
        }
    }
}
```

Counterexample: 95009

αν το  $p$  δεν  
είναι πρώτος,  
προχώρησε  
στο επόμενο  $p$

μόλις βρεις  
αντιπαράδειγμα  
σταμάτησε

$$17 \times 95,009 = 1,615,153 = 383 \times 4217 + 42$$

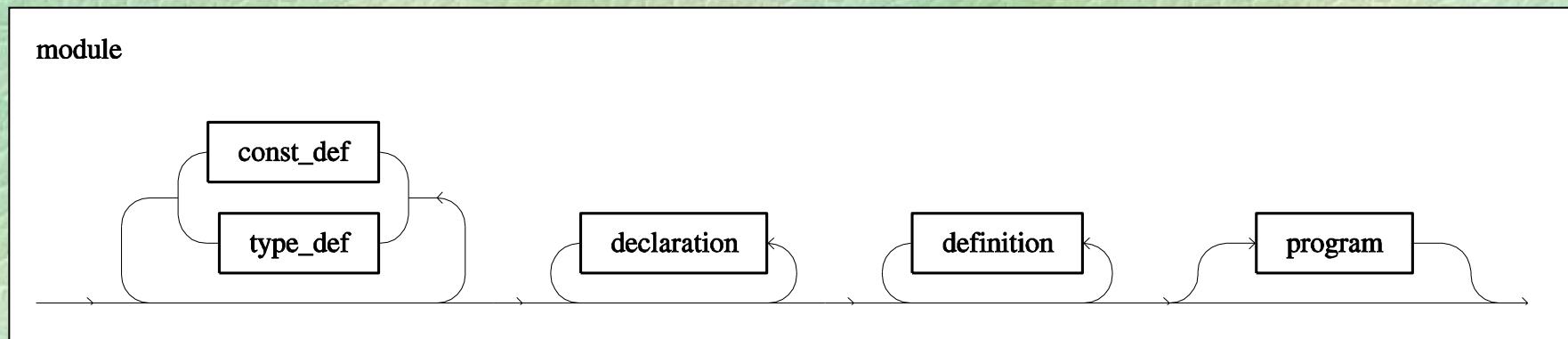
# Κενή εντολή

- Συμβολίζεται με ένα semicolon
- Δεν κάνει τίποτα όταν εκτελείται
- Παράδειγμα

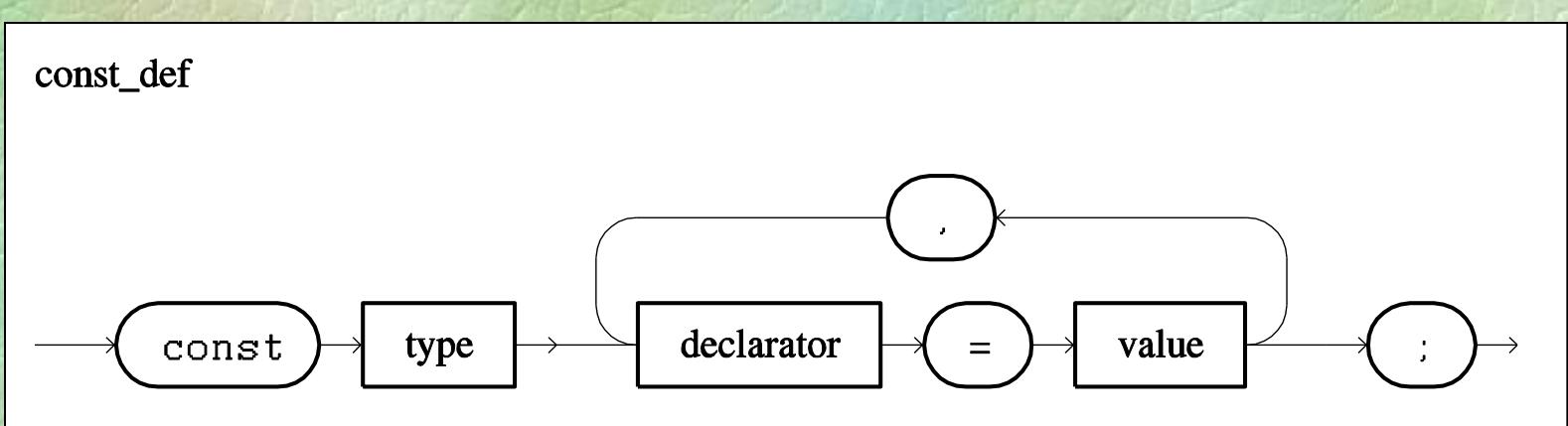
```
if (x > 4) {  
    y = 1;  
    x = x-5;  
    ;           // κενή εντολή  
}
```

# Δομή του προγράμματος, ξανά

- Μονάδα κώδικα **module**
  - βρίσκεται σε ξεχωριστό αρχείο προγράμματος
- Αποτελείται από:
  - δηλώσεις σταθερών και τύπων
  - δηλώσεις και ορισμούς υποπρογραμμάτων
  - τον ορισμό ενός (απλού) προγράμματος



- Σαν μεταβλητές, αλλά:
  - προηγείται η λέξη-κλειδί **const**
  - υποχρεωτική αρχικοποίηση
  - απαγορεύεται η ανάθεση



## □ Παραδείγματα

```
const int N = 100000;  
const REAL pi = 3.1415926535,  
          e = 2.7182818284;  
const char SPACE = ' ';
```

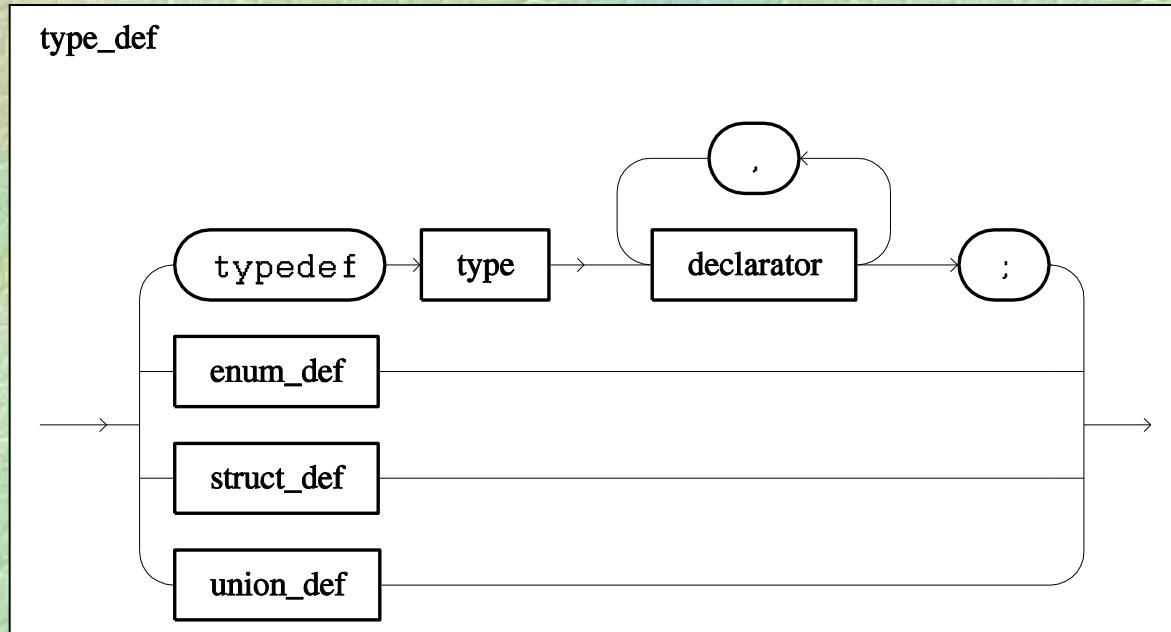
## □ Χρήση αντί των σταθερών εκφράσεων

□ π.χ. **for (int i=0; i<N; i++) ...**

## □ Προκαθορισμένες σταθερές

□ π.χ. **INT\_MIN, INT\_MAX**

- Σαν δηλώσεις μεταβλητών, αλλά:
  - προηγείται η λέξη-κλειδί **typedef**
  - όχι αρχικοποίηση
  - δηλώνουν ονόματα τύπων, όχι μεταβλητών



- Παραδείγματα

```
typedef int number;  
typedef bool bit;  
typedef REAL real;
```

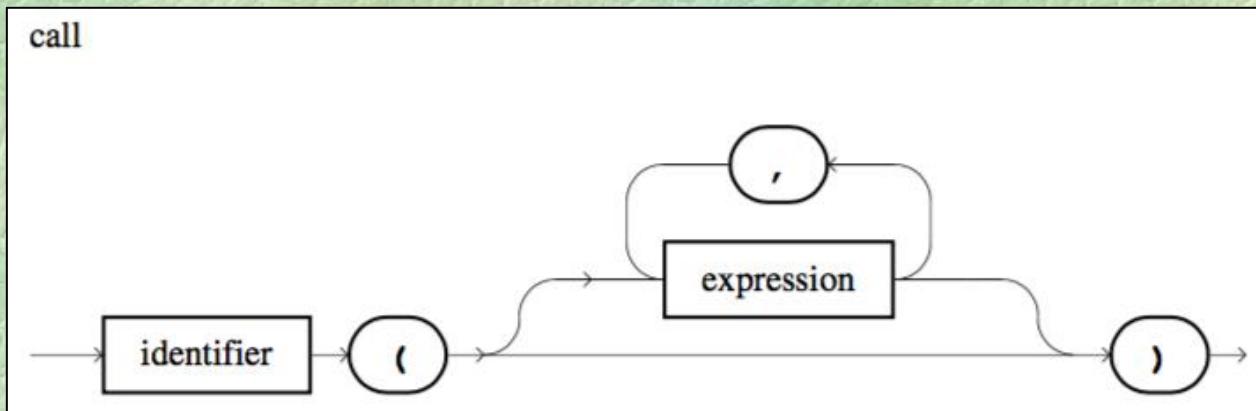
- Χρήση αντί των τύπων

```
number n;  
bit b;    real r;
```

- Προκαθορισμένοι τύποι

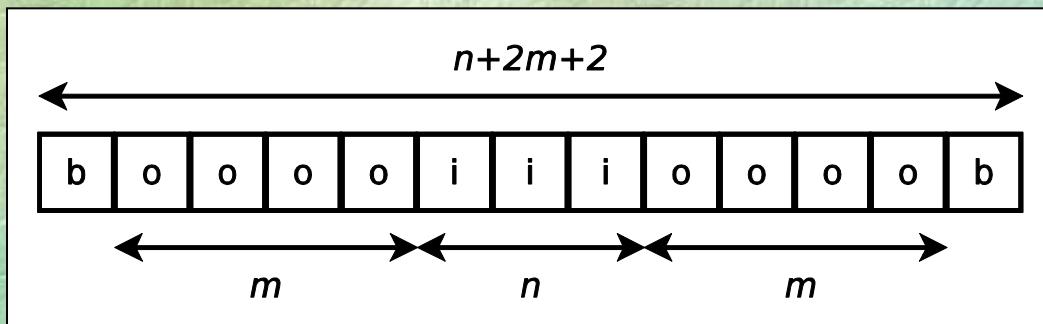
- π.χ. **int, REAL, bool, char**

- Ορίζονται στο τμήμα δηλώσεων
- Κάθε ορισμός διαδικασίας περιέχει:
  - την επικεφαλίδα της
  - το σώμα της
- Καλούνται με αναγραφή του ονόματός τους και απαρίθμηση των παραμέτρων

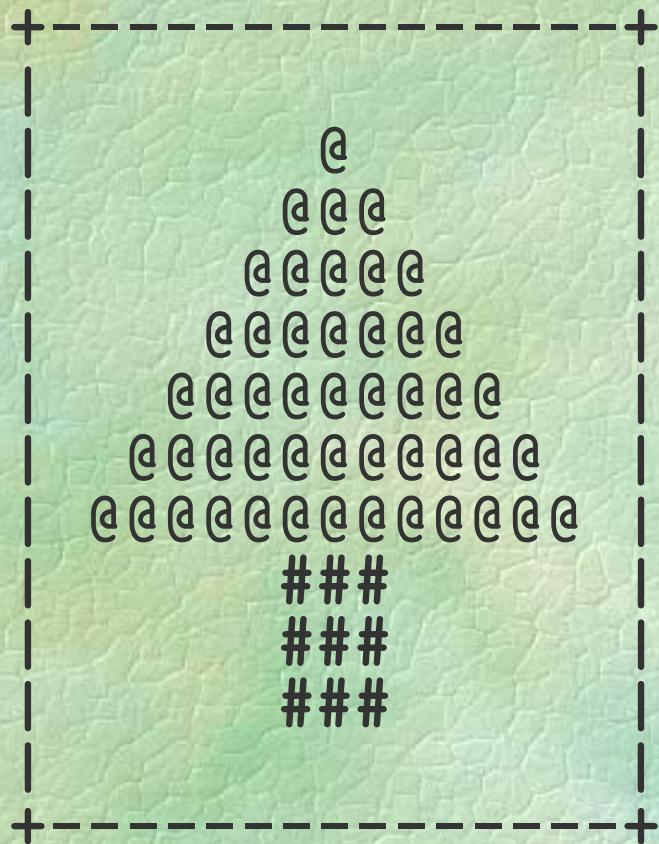


- Τυπικές (formal) παράμετροι ενός υποπρογράμματος είναι αυτές που ορίζονται στην επικεφαλίδα του
- Πραγματικές (actual) παράμετροι ενός υποπρογράμματος είναι αυτές που δίνονται κατά την κλήση του
- Σε κάθε κλήση, οι πραγματικές παράμετροι πρέπει να αντιστοιχούν μία προς μία στη σειρά και στον τύπο με τις τυπικές

- Χριστουγεννιάτικη καρτ ποστάλ
- Κάθε γραμμή έχει τη μορφή:



- **b, i, o** : áκρα, μέσο και υπόλοιπο γραμμής
- **n, m** : διαστάσεις



```
void line (char border, int n, char inside,
           int m, char outside)
```

{

```
int i;
```

τοπική μεταβλητή

```
WRITE(border); // αριστερό πλαίσιο
for (i=0; i<m; i++) WRITE(outside);
for (i=0; i<n; i++) WRITE(inside);
for (i=0; i<m; i++) WRITE(outside);
WRITELN(border); // δεξιό πλαίσιο
```

}

όνομα διαδικασίας

εμβέλεια  
του i

τυπικές  
παράμετροι

```
void line (char border, int n, char inside,
            int m, char outside)
{ ... }
```

```
PROGRAM { // tree_postcard
    int i;
    line('+', 15, '-', 0, ' ') ; // πάνω πλαισιο
    line('|', 15, ' ', 0, ' ') ;
    for (i=1; i<=13; i+=2)
        line('|', i, '@', (15-i)/2, ' ') ;
    for (i=1; i<=3; i++)
        line('|', 3, '#', 6, ' ') ;
    line('|', 15, ' ', 0, ' ') ;
    line('+', 15, '-', 0, ' ') ; // κάτω πλαισιο
}
```

πραγματικές  
παράμετροι

- Τυπικές (formal) παράμετροι ενός υποπρογράμματος είναι αυτές που ορίζονται στην επικεφαλίδα του
- Πραγματικές (actual) παράμετροι ενός υποπρογράμματος είναι αυτές που δίνονται κατά την κλήση του
- Σε κάθε κλήση, οι πραγματικές παράμετροι πρέπει να αντιστοιχούν μία προς μία στη σειρά και στον τύπο με τις τυπικές

```
void line (char border, int n, char inside,
           int m, char outside)
```

{

```
int i;
```

τοπική μεταβλητή

```
WRITE(border); // αριστερό πλαίσιο
for (i=0; i<m; i++) WRITE(outside);
for (i=0; i<n; i++) WRITE(inside);
for (i=0; i<m; i++) WRITE(outside);
WRITELN(border); // δεξιό πλαίσιο
```

}

όνομα διαδικασίας

εμβέλεια  
του i

τυπικές  
παράμετροι

- Εμβέλεια ενός ονόματος (π.χ. μεταβλητής) είναι το τμήμα του προγράμματος όπου επιτρέπεται η χρήση του
- Τοπικά (local) ονόματα είναι αυτά που δηλώνονται σε ένα υποπρόγραμμα
- Γενικά (global) ονόματα είναι αυτά που δηλώνονται έξω από υποπρόγραμμα και έχουν εμβέλεια σε ολόκληρο το module

## □ Σύγκρουση ονομάτων

- όταν μία παράμετρος ή τοπική μεταβλητή έχει ένα όνομα που χρησιμοποιείται ήδη σε εξωτερικότερη εμβέλεια
- το όνομα στο εσωτερικότερο block **κρύβει** αυτό στο εξωτερικότερο block

## □ Εκτέλεση με το χέρι

## □ Trace tables

```
int a, b, c;

void p42 (int y, int b) {
    int c = 42; WRITELN(a, b, c, y);
    a = a + b; c = c + 1; b = c + b; y = y-1;
    WRITELN(a, b, c, y);
}

void p17 (int a, int x) {
    int b = 17; WRITELN(a, b, c, x);
    p42(b, x); WRITELN(a, b, c, x);
}

PROGRAM { // proc_example
    a = 1; b = 2; c = 3; p17(b, c); p42(c, a);
}
```

| Global | a  | b  | c  |
|--------|----|----|----|
|        | 1  | 2  | 3  |
|        | 4  |    |    |
|        | 8  |    |    |
| p17    | a  | x  | b  |
|        | 2  | 3  | 17 |
| p42    | y  | b  | c  |
|        | 17 | 3  | 42 |
|        | 16 | 46 | 43 |
| p42    | y  | b  | c  |
|        | 3  | 4  | 42 |
|        | 2  | 47 | 43 |

Output

2 17 3 3  
 1 3 42 17  
 4 46 43 16  
 2 17 3 3  
 4 4 42 3  
 8 47 43 2

```
int a, b, c;
```

```
PROGRAM { // proc_example
    a = 1; b = 2; c = 3; p17(b, c); p42(c, a);
}
```

| Global | a | b | c |
|--------|---|---|---|
| 1      | 2 | 3 |   |
| 4      |   |   |   |
| 8      |   |   |   |

| p17 | a | x | b  |
|-----|---|---|----|
|     | 2 | 3 | 17 |

| p42 | y  | b  | c  |
|-----|----|----|----|
|     | 17 | 3  | 42 |
|     | 16 | 46 | 43 |

| p42 | y | b  | c  |
|-----|---|----|----|
|     | 3 | 4  | 42 |
|     | 2 | 47 | 43 |

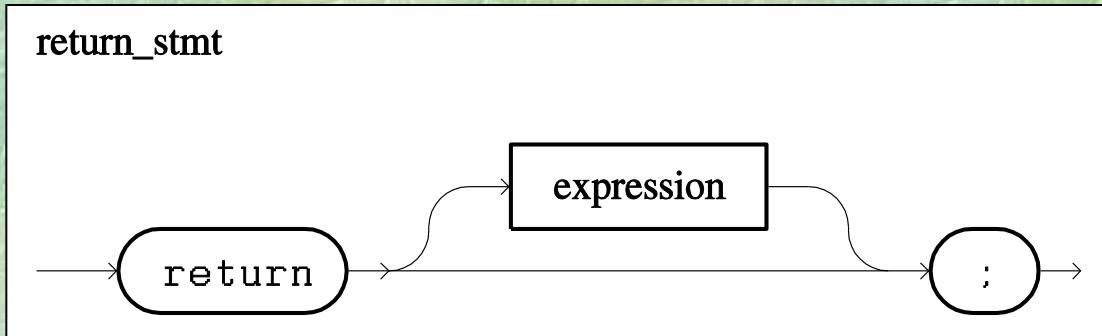
```
void p17 (int a, int x) {
    int b = 17;
    WRITELN(a, b, c, x);
    p42(b, x);
    WRITELN(a, b, c, x);
}
```

Output

```
2 17 3 3
1 3 42 17
4 46 43 16
2 17 3 3
4 4 42 3
8 47 43 2
```

```
void p42 (int y, int b) {
    int c = 42;
    WRITELN(a, b, c, y);
    a = a + b;
    c = c + 1;
    b = c + b;
    y = y - 1;
    WRITELN(a, b, c, y);
}
```

- Όπως οι διαδικασίες, αλλά επιστρέφουν μια τιμή ως **αποτέλεσμα**
- Δεν μπορούν να χρησιμοποιηθούν ως εντολές αλλά μόνο σε παραστάσεις
- Επιστροφή αποτελέσματος με την εντολή **return** (και χωρίς παράσταση, για διαδικασίες)



```
int gcd (int a, int b) {  
    a = abs(a); b = abs(b);  
    while (a > 0 && b > 0)  
        if (a > b) a = a % b;  
        else         b = b % a;  
    return a+b;  
}
```

```
PROGRAM { // gcd_func  
    int x, y;  
  
    WRITE("Give x: "); x = READ_INT();  
    WRITE("Give y: "); y = READ_INT();  
    WRITELN("gcd is:", gcd(x, y));  
}
```

# Δομημένος προγραμματισμός

- **Ιδέα:** κάθε ανεξάρτητη λειτουργία του προγράμματος πρέπει να αντιστοιχεί σε ανεξάρτητο υποπρόγραμμα
- **Πλεονεκτήματα**
  - Ευκολότερη ανάπτυξη προγραμμάτων («διαιρεί και βασίλευε»)
  - Ευκολότερη ανίχνευση σφαλμάτων
  - Επαναχρησιμοποίηση έτοιμων υποπρογραμμάτων

# Βαθιαία συγκεκριμενοποίηση

- Περιγραφή επίλυσης προβλήματος
  - Εισαγωγή και αποθήκευση δεδομένων
    - τρόπος εισαγωγής δεδομένων
    - έλεγχος ορθότητας δεδομένων
  - Αλγόριθμος επεξεργασίας
    - περιγραφή του αλγορίθμου
    - κωδικοποίηση στη γλώσσα προγραμματισμού
  - Παρουσίαση αποτελεσμάτων
    - τρόπος και μορφή παρουσίασης αποτελεσμάτων

# Παρουσίαση και συντήρηση (i)

- Ποιοτικά χαρακτηριστικά προγραμμάτων
  - Αναγνωσιμότητα
    - απλότητα
    - κατάλληλη επιλογή ονομάτων, π.χ.  
**monthly\_income      incomeBeforeTaxes**
    - στοίχιση
    - σχόλια
  - Φιλικότητα προς το χρήστη
  - Τεκμηρίωση
  - Συντήρηση
  - Ενημέρωση

# Παρουσίαση και συντήρηση (ii)

## □ Στοίχιση

- Πρόγραμμα και υποπρογράμματα

```
PROGRAM {      int f(int x) {  
    δηλώσεις  
    εντολές  
}  
    δηλώσεις  
    εντολές  
}
```

- Απλές εντολές

```
if (...) εντολή  
else εντολή
```

```
while (...) εντολή  
for (...) εντολή
```

# Παρουσίαση και συντήρηση (iii)

## □ Στοίχιση (συνέχεια)

### □ Σύνθετες εντολές

```
if (...) {      while (...) {      for (...) {  
    εντολές          εντολές          εντολές  
}                  }                  }  
else {  
  εντολές  
}  
  
do {  
  εντολές  
} while (...);
```

# Παρουσίαση και συντήρηση (iv)

## □ Στοίχιση (συνέχεια)

### □ Σύνθετες εντολές (συνέχεια)

```
switch (...) {  
    case τιμή1 : εντολές1  
    case τιμή2 : εντολές2  
    ...  
    case τιμήn : εντολέςn  
    default : εντολέςn+1  
}
```

# Έξοδος με μορφοποίηση

## □ Ακέραιες τιμές

► **WRITELN (FORM (42 , 4)) ;**



4 2

## □ ... αλλά και οτιδήποτε άλλο

**WRITELN (FORM ("hello" , 8)) ;**



h e l l o

## □ Πραγματικές τιμές

**WRITELN (FORM (3.1415926, 8, 4)) ;**



3 . 1 4 1 6 9

# Αρχεία κειμένου

## □ Ανακατεύθυνση εισόδου και εξόδου

```
PROGRAM { // redirection
    int n, i, sum = 0;

    INPUT ("file-to-read-from.txt");
    OUTPUT ("file-to-write-to.txt");

    n = READ_INT();
    for (i=0; i<n; i++)
        sum = sum + READ_INT();
    WRITELN(sum);
}
```

# Τακτικοί τύποι

- Οι τύποι **int**, **bool** και **char**
- Απαριθμητοί τύποι

```
enum color {white, red, blue, green,  
            yellow, black, purple};  
enum sex     {male, female};  
enum day     {mon, tue, wed, thu,  
              fri, sat, sun};  
  
color c = green;  
day d = fri;
```

- Πράξεις με τακτικούς τύπους
  - τελεστές σύγκρισης ==, !=, <, >, <=, >=

- Δομημένη μεταβλητή: αποθηκεύει μια συλλογή από τιμές δεδομένων
- Πίνακας (array): δομημένη μεταβλητή που αποθηκεύει πολλές τιμές του ίδιου τύπου

`int n[5];`

ορίζει έναν πίνακα πέντε ακεραίων, τα στοιχεία του οποίου είναι:

`n[0], n[1], n[2], n[3], n[4]`

και έχουν τύπο `int`

### □ Παραδείγματα

```
REAL a[10];  
int b[20];  
char c[30];  
  
...  
  
a[1] = 4.2;  
a[3] = READ_REAL();  
a[9] = a[1];  
  
b[2] = b[2]+1;  
  
c[26] = 't';
```

### □ Διάβασμα ενός πίνακα

- γνωστό μέγεθος

```
for (int i=0; i<10; i++)
    a[i] = READ_INT();
```

- πρώτα διαβάζεται το μέγεθος

```
int n = READ_INT();
for (int i=0; i<n; i++)
    a[i] = READ_INT();
```

- στα παραπάνω πρέπει να προηγηθεί

```
int a[100]; // κάτι όχι μικρότερο του 10
```

## □ Διάβασμα ενός πίνακα (συνέχεια)

- τερματισμός με την τιμή 0 (φρουρός/sentinel)  

```
int x = READ_INT(), i=0;
while (x != 0) {
    a[i] = x; i = i+1; x = READ_INT();
}
```
- στο παραπάνω πρέπει να προηγηθεί  

```
int a[100];
```
- **Προσοχή:** δε γίνεται έλεγχος για το πλήθος των στοιχείων που δίνονται!

# Πράξεις με πίνακες

- Απλές πράξεις, π.χ.

**a[k] = a[k]+1;**

**a[k] = a[1]+a[n];**

**for (int i=0; i<10; i++)**

**WRITELN(a[i]);**

**if (a[k] > a[k+1]) ...**

- Αρχικοποίηση (με μηδενικά)

**for (int i=0; i<10; i++) a[i]=0;**

- Εύρεση ελάχιστου στοιχείου

**x = a[0];**

**for (int i=1; i<10; i++)**

**if (a[i] < x) x = a[i];**

# Γραμμική αναζήτηση (i)

- Πρόβλημα (αναζήτησης): δίνεται ένας πίνακας ακεραίων **a** και ζητείται να βρεθεί αν υπάρχει ο ακέραιος **x** στα στοιχεία του

```
PROGRAM { // linsearch
    int x, n, a[100];
    άλλες δηλώσεις;
    τίτλος επικεφαλίδα;
    οδηγίες στο χρήστη;
    x = READ_INT();
    διάβασμα του πίνακα;
    ψάξιμο στον πίνακα για τον x;
    παρουσίαση αποτελεσμάτων
}
```

# Γραμμική αναζήτηση (ii)

## □ Μια δυνατή συγκεκριμενοποίηση

```
n = READ_INT();  
for (i=0; i<n; i++) a[i] = READ_INT();  
i=0;  
while (i < n && a[i] != x) i=i+1;  
if (i < n)  
    WRITELN("Το βρήκα στη θέση", i);  
else  
    WRITELN("Δεν το βρήκα");
```

- Στη χειρότερη περίπτωση θα ελεγχθούν όλα τα στοιχεία του πίνακα
- Απαιτούνται  $a n + b$  βήματα  $\Rightarrow$  γραμμική ( $a, b$  σταθερές,  $n$  το μέγεθος του πίνακα)

# Γραμμική αναζήτηση

## (iii)

### □ Εναλλακτική συγκεκριμενοποίηση #1

```
i = 0;  
do  
    if (a[i] == x) break; else i = i+1;  
while (i < n);  
  
if (i < n)  
    WRITELN("Το βρήκα στη θέση", i);  
else  
    WRITELN("Δεν το βρήκα");
```

# Γραμμική αναζήτηση

## (iv)

### □ Εναλλακτική συγκεκριμενοποίηση #2

```
i = 0;  
do  
    if (a[i] == x) found = true;  
    else { found = false; i = i+1; }  
while (!found && i < n);  
  
if (found)  
    WRITELN("Το βρήκα στη θέση", i);  
else  
    WRITELN("Δεν το βρήκα");
```

# Γραμμική αναζήτηση (v)

## □ Εναλλακτική συγκεκριμενοποίηση #3

```
i = 0; found = false;  
do  
    if (a[i] == x) found = true;  
    else i = i+1;  
while (!found && i < n);  
  
if (found)  
    WRITELN("Το βρήκα στη θέση", i);  
else  
    WRITELN("Δεν το βρήκα");
```

## □ Εναλλακτική συγκεκριμενοποίηση #4

```
i = 0;  
do {  
    found = a[i] == x;  
    i = i+1;  
} while (!found && i < n);  
  
if (found)  
    WRITELN("Το βρήκα στη θέση", i-1);  
else  
    WRITELN("Δεν το βρήκα");
```

- Προϋπόθεση: ο πίνακας να είναι ταξινομημένος, π.χ. σε αύξουσα διάταξη
- Είναι πολύ πιο αποδοτική από τη γραμμική αναζήτηση
  - Στη χειρότερη περίπτωση απαιτούνται  $a \log_2 n + b$  βήματα ( $a, b$  σταθερές,  $n$  το μέγεθος του πίνακα)

## □ Το πρόγραμμα

```
const int N = 100;  
  
PROGRAM { // binsearch  
    int i, x, n, first, last, mid, a[N];  
    Mήνυμα επικεφαλίδα και οδηγίες χρήσης;  
    n = READ_INT(); // κατά αύξουσα σειρά  
    for (i=0; i<n; i++)  
        a[i] = READ_INT();  
    x = READ_INT();  
    Αναζήτηση και εμφάνιση αποτελέσματος  
}
```

## □ Αναζήτηση και εμφάνιση αποτελέσματος

```
first = 0; last = n-1;  
while (first <= last) {  
    mid = (first + last) / 2;  
    if (x < a[mid]) last = mid-1;  
    else if (x > a[mid]) first = mid+1;  
    else break;  
}  
if (first <= last)  
    WRITELN("Το βρήκα στη θέση", mid);  
else  
    WRITELN("Δεν το βρήκα");
```

# Πολυδιάστατοι πίνακες

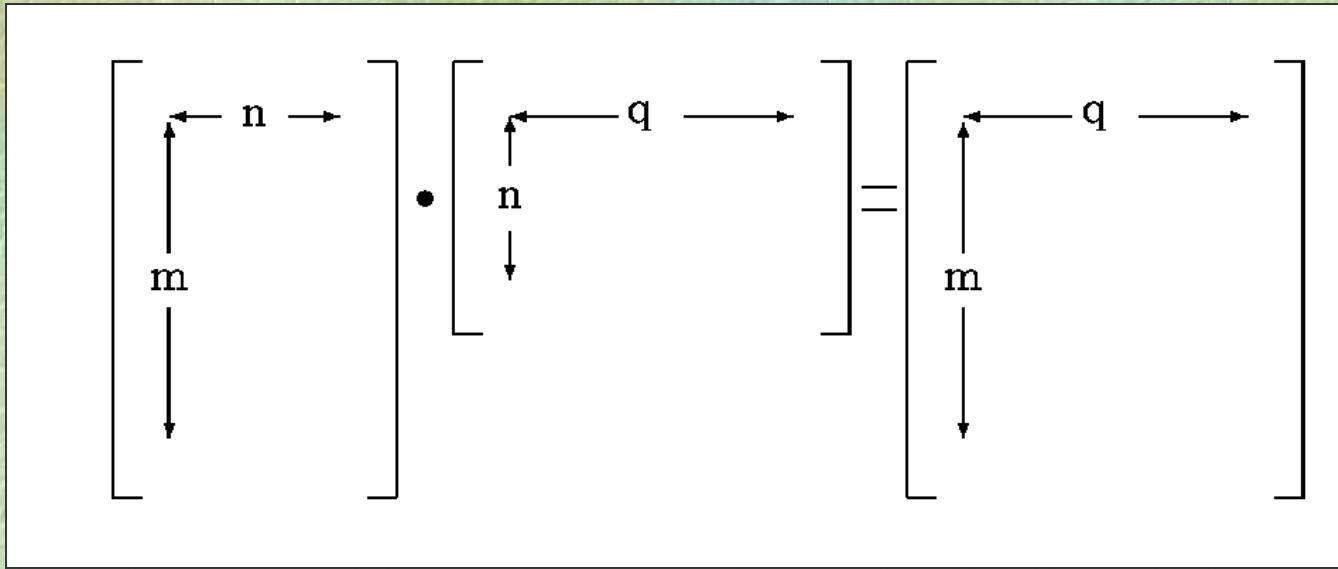
## □ Παράδειγμα

```
int a[10][16];  
  
...  
a[1][13] = 42;  
  
...  
for (i=0; i<10; i++)  
  for (j=0; j<16; j++)  
    a[i][j] = READ_INT();
```

# Πολλαπλασιασμός πινάκων (i)

- Δίνονται οι πίνακες:  $a$  ( $m \times n$ ),  $b$  ( $n \times q$ )
- Ζητείται ο πίνακας:  $c = a \cdot b$  ( $m \times q$ ) όπου:

$$c_{i,j} = \sum_{k=1}^n a_{i,k} b_{k,j}$$



# Πολλαπλασιασμός πινάκων (ii)

## □ Το πρόγραμμα

```
REAL a[m] [n] , b[n] [q] , c[m] [q] ;  
...  
for (i=0; i<m; i++)  
  for (j=0; j<q; j++) {  
    c[i][j] = 0;  
    for (k=0; k<n; k++)  
      c[i][j] = c[i][j] +  
        a[i][k]*b[k][j];  
  }
```

- Διδιάστατοι πίνακες ( $n \times n$ ) που περιέχουν όλους τους φυσικούς μεταξύ 0 και  $n^2 - 1$ 
  - το άθροισμα των στοιχείων κάθε στήλης, γραμμής και διαγωνίου είναι σταθερό
- Πρόβλημα: κατασκευή μαγικού τετραγώνου ( $n \times n$ ) για περιττό  $n$

|    |    |    |    |    |
|----|----|----|----|----|
| 10 | 9  | 3  | 22 | 16 |
| 17 | 11 | 5  | 4  | 23 |
| 24 | 18 | 12 | 6  | 0  |
| 1  | 20 | 19 | 13 | 7  |
| 8  | 2  | 21 | 15 | 14 |

# Μαγικά τετράγωνα

(ii)

|   |  |  |  |  |  |  |  |  |
|---|--|--|--|--|--|--|--|--|
| 0 |  |  |  |  |  |  |  |  |
|   |  |  |  |  |  |  |  |  |
|   |  |  |  |  |  |  |  |  |
|   |  |  |  |  |  |  |  |  |
|   |  |  |  |  |  |  |  |  |

|   |  |  |  |  |  |  |  |  |
|---|--|--|--|--|--|--|--|--|
| 0 |  |  |  |  |  |  |  |  |
|   |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |
|   |  |  |  |  |  |  |  |  |
|   |  |  |  |  |  |  |  |  |

|   |   |  |  |  |  |  |  |  |
|---|---|--|--|--|--|--|--|--|
| 0 |   |  |  |  |  |  |  |  |
|   |   |  |  |  |  |  |  |  |
| 1 |   |  |  |  |  |  |  |  |
|   |   |  |  |  |  |  |  |  |
|   | 2 |  |  |  |  |  |  |  |

|   |   |  |  |  |  |  |  |  |
|---|---|--|--|--|--|--|--|--|
| 3 |   |  |  |  |  |  |  |  |
|   |   |  |  |  |  |  |  |  |
| 1 |   |  |  |  |  |  |  |  |
|   | 2 |  |  |  |  |  |  |  |
|   |   |  |  |  |  |  |  |  |

|   |   |  |  |  |  |  |  |  |
|---|---|--|--|--|--|--|--|--|
| 3 |   |  |  |  |  |  |  |  |
|   | 4 |  |  |  |  |  |  |  |
|   |   |  |  |  |  |  |  |  |
| 1 |   |  |  |  |  |  |  |  |
| 2 |   |  |  |  |  |  |  |  |

|   |   |   |  |  |  |  |  |  |
|---|---|---|--|--|--|--|--|--|
| 3 |   |   |  |  |  |  |  |  |
|   | 5 | 4 |  |  |  |  |  |  |
|   |   |   |  |  |  |  |  |  |
| 1 |   |   |  |  |  |  |  |  |
| 2 |   |   |  |  |  |  |  |  |

|   |   |   |  |  |  |  |  |  |
|---|---|---|--|--|--|--|--|--|
| 3 |   |   |  |  |  |  |  |  |
|   | 5 | 4 |  |  |  |  |  |  |
|   |   |   |  |  |  |  |  |  |
| 1 |   |   |  |  |  |  |  |  |
| 2 |   |   |  |  |  |  |  |  |

|   |   |   |  |  |  |  |  |  |
|---|---|---|--|--|--|--|--|--|
| 3 |   |   |  |  |  |  |  |  |
|   | 5 | 4 |  |  |  |  |  |  |
|   |   |   |  |  |  |  |  |  |
| 1 |   |   |  |  |  |  |  |  |
| 2 |   |   |  |  |  |  |  |  |

|   |   |   |  |  |  |  |  |  |
|---|---|---|--|--|--|--|--|--|
| 3 |   |   |  |  |  |  |  |  |
|   | 5 | 4 |  |  |  |  |  |  |
|   |   |   |  |  |  |  |  |  |
| 1 |   |   |  |  |  |  |  |  |
| 2 |   |   |  |  |  |  |  |  |

|   |   |   |  |  |  |  |  |  |
|---|---|---|--|--|--|--|--|--|
| 9 | 3 |   |  |  |  |  |  |  |
|   | 5 | 4 |  |  |  |  |  |  |
|   |   |   |  |  |  |  |  |  |
| 1 |   |   |  |  |  |  |  |  |
| 8 | 2 |   |  |  |  |  |  |  |

|    |   |   |  |  |  |  |  |  |
|----|---|---|--|--|--|--|--|--|
| 10 | 9 | 3 |  |  |  |  |  |  |
|    | 5 | 4 |  |  |  |  |  |  |
|    |   |   |  |  |  |  |  |  |
| 1  |   |   |  |  |  |  |  |  |
| 8  | 2 |   |  |  |  |  |  |  |

|    |    |   |   |  |  |  |  |  |
|----|----|---|---|--|--|--|--|--|
| 10 | 9  | 3 |   |  |  |  |  |  |
|    | 11 | 5 | 4 |  |  |  |  |  |
|    |    |   |   |  |  |  |  |  |
| 1  |    |   |   |  |  |  |  |  |
| 8  | 2  |   |   |  |  |  |  |  |

### □ Κατασκευή για περιπτώ $n$

```
int a[17][17], n = 5;
int i = n/2, j = n, k = 0;
for (int h=0; h<n; h++) {
    j = j-1; a[i][j] = k; k = k+1;
    for (int m=1; m<n; m++) {
        j = (j+1) % n; i = (i+1) % n;
        a[i][j] = k; k = k+1;
    }
}

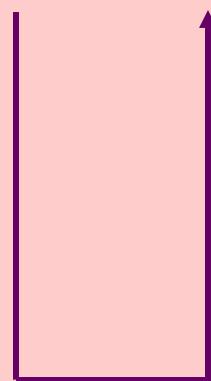
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++)
        WRITE(FORM(a[i][j], 4));
    WRITELN();
}
```

- Αναδρομικές διαδικασίες ή συναρτήσεις:  
αυτές που καλούν τον εαυτό τους
- Το αρχικό πρόβλημα ανάγεται στην  
επίλυση ενός ή περισσότερων μικρότερων  
προβλημάτων του ίδιου τύπου
- Παράδειγμα: παραγοντικό
  - $n! = n * (n-1) * (n-2) * \dots * 2 * 1$
  - Αναδρομικός ορισμός
$$0! = 1 \quad (n+1)! = (n+1) * n!$$

## □ Παράδειγμα: παραγοντικό (συνέχεια)

```
int fact (int n) {  
    if (n==0) return 1;  
    else        return fact(n-1) * n;  
}
```

πρόγραμμα καλεί **fact(3)**  
**fact(3)** καλεί **fact(2)**  
**fact(2)** καλεί **fact(1)**  
**fact(1)** καλεί **fact(0)**  
**fact(0)**



συνεχίζει...  
επιστρέφει 6  
επιστρέφει 2  
επιστρέφει 1  
επιστρέφει 1

# Αναδρομή

(iii)



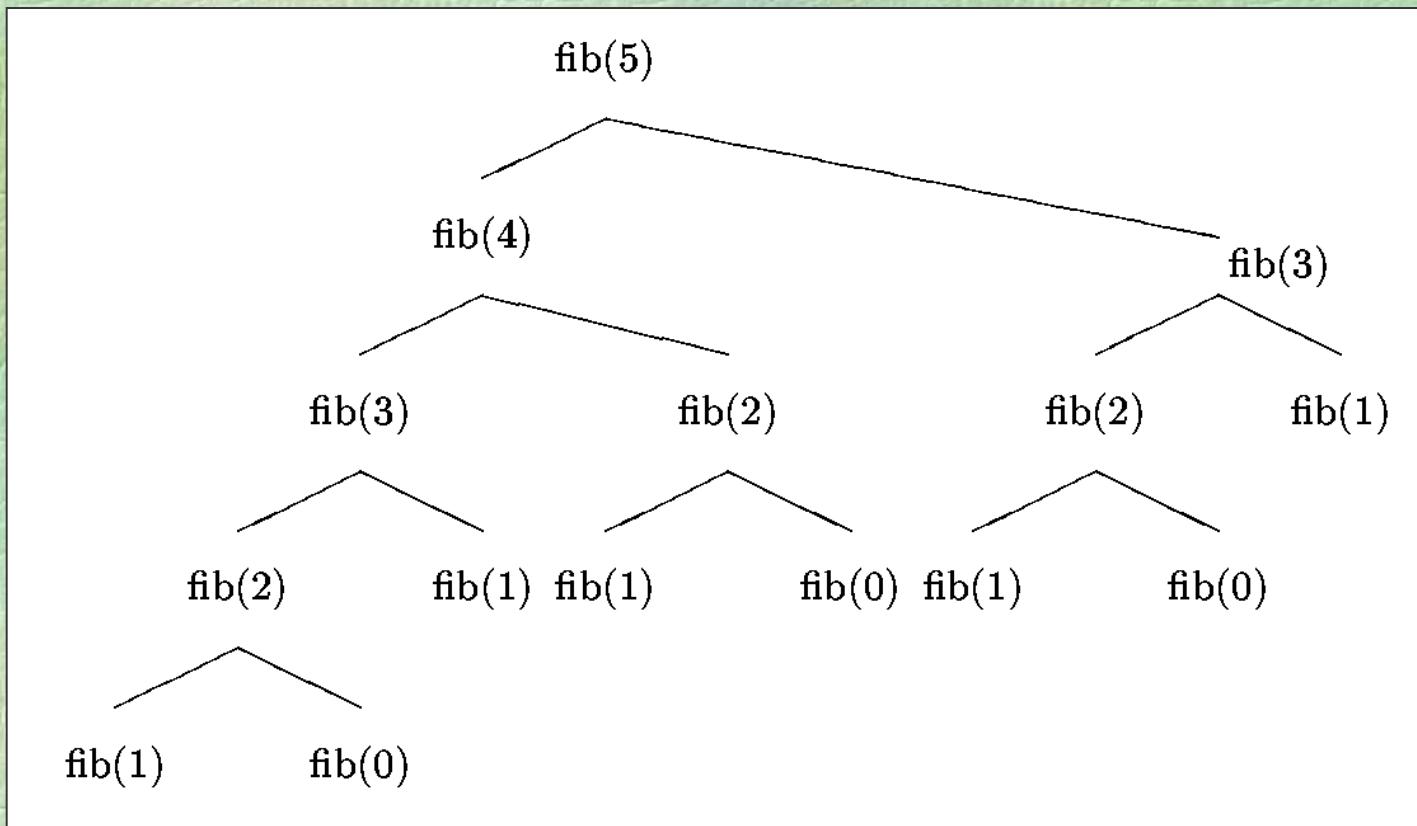
## □ Αριθμοί Fibonacci

- $F_0 = 0$  ,  $F_1 = 1$
- $F_{n+2} = F_n + F_{n+1}$  ,  $\forall n \in \mathbf{N}$

## □ Αναδρομική συνάρτηση υπολογισμού

```
int fib (int n) {  
    if (n==0 || n==1)  
        return n;  
    else  
        return fib(n-1) + fib(n-2);  
}
```

- Αυτός ο αναδρομικός υπολογισμός των αριθμών Fibonacci δεν είναι αποδοτικός



## □ Μέγιστος κοινός διαιρέτης

- Αναδρομική υλοποίηση του αλγορίθμου του Ευκλείδη

```
int gcd (int i, int j) {  
    if (i==0 || j==0)  
        return i+j;  
    else if (i > j)  
        return gcd(i%j, j);  
    else  
        return gcd(j%i, i);  
}
```

□ Ύψωση σε δύναμη:  $x^n = x$  οτιδήποτε,  $n \in \mathbf{N}$

- με διαδοχικούς πολλαπλασιασμούς —  $O(n)$
- με διαδοχικό τετραγωνισμό —  $O(?)$

$$x^{2k} = (x^2)^k$$

$$x^{2k+1} = x \cdot (x^2)^k$$

$$x^0 = 1$$

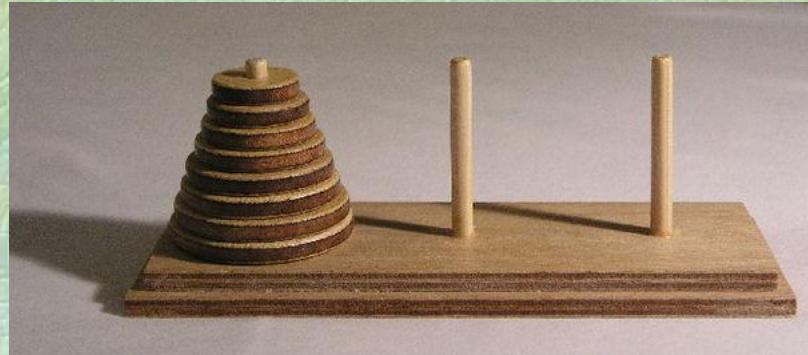
□ Ύψωση σε δύναμη:  $x^n = x$  οτιδήποτε,  $n \in \mathbf{N}$

- με διαδοχικό τετραγωνισμό

```
int pow (int x, int n) {  
    if (n == 0)  
        return 1;  
    else if (n%2 == 0)  
        return pow(x*x, n/2);  
    else  
        return x * pow(x*x, n/2);  
}
```

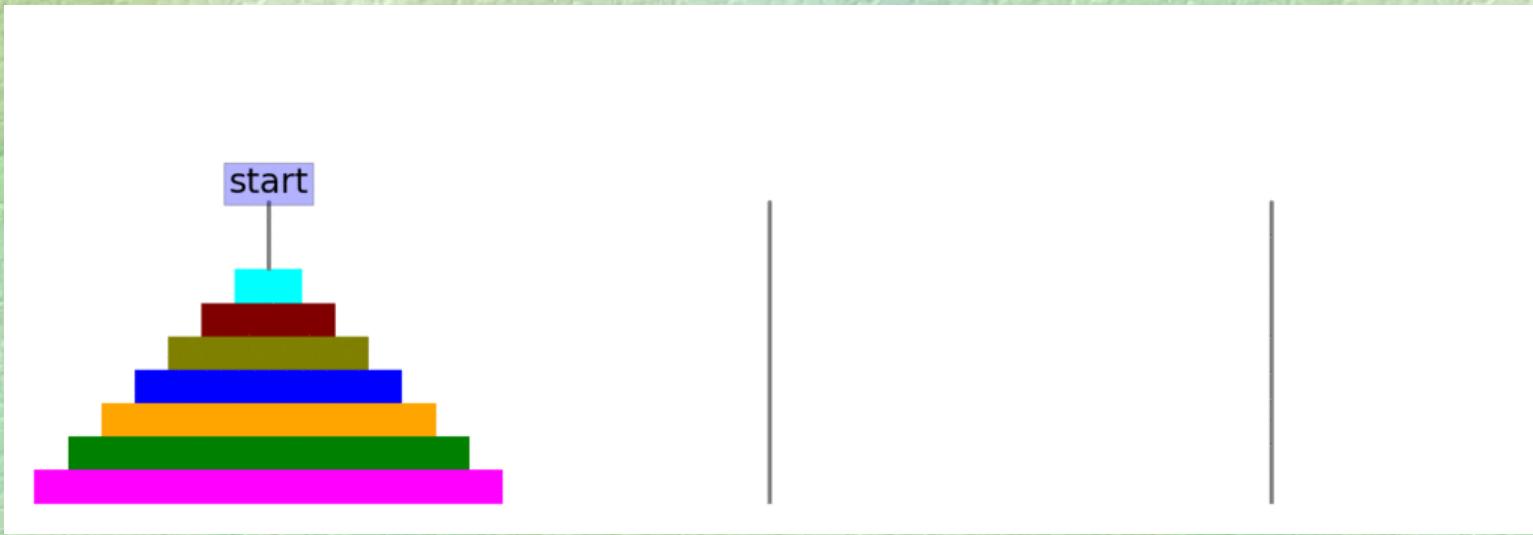
Πολυπλοκότητα;  $O(\log n)$

## Οι πύργοι του Hanoi

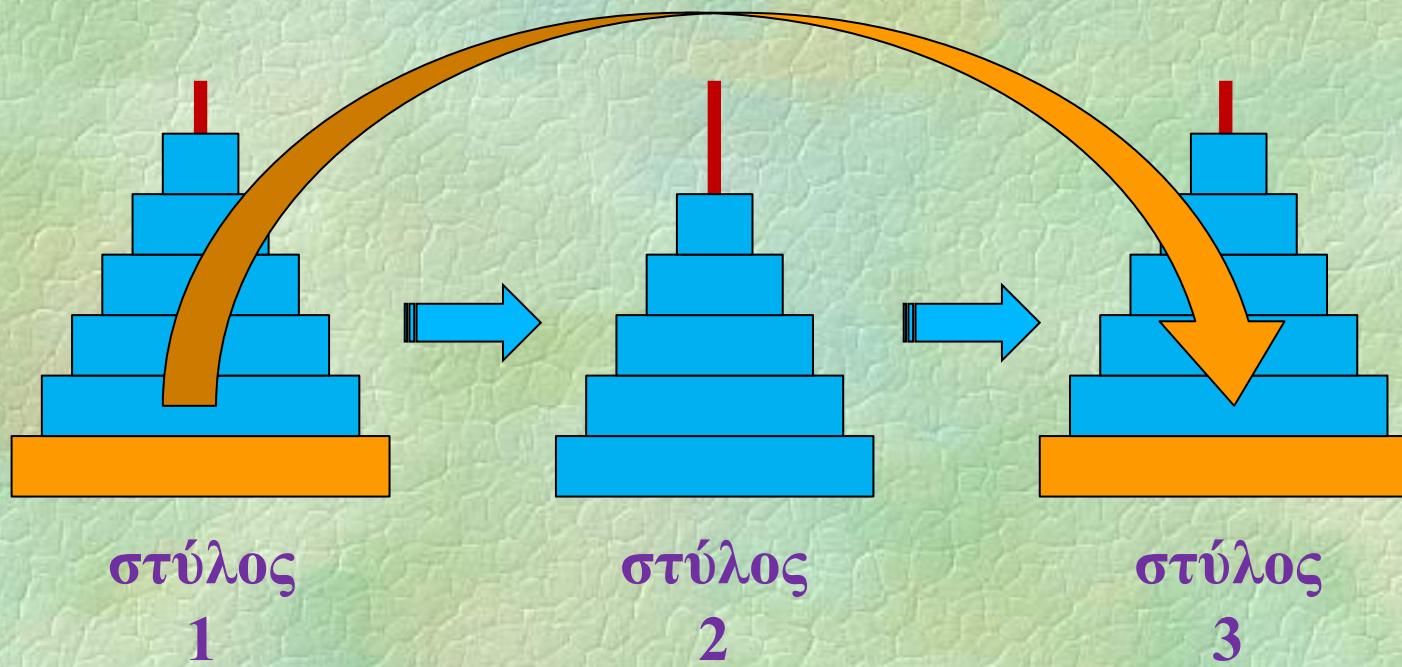


- **Ζητούμενο:** Να μεταφερθούν οι δίσκοι από τον αριστερό στύλο στον δεξιό
  - μόνο ένας δίσκος μεταφέρεται κάθε φορά
  - από την κορυφή ενός στύλου στην κορυφή κάποιου άλλου στύλου
  - απαγορεύεται να τοποθετηθεί μεγαλύτερος δίσκος πάνω σε μικρότερο

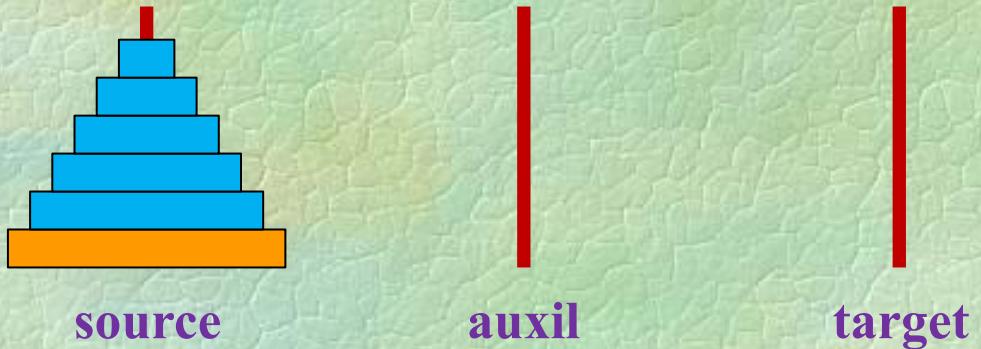
- Οι πύργοι του Hanoi
- Λύση για 7 δίσκους με 127 κινήσεις



## □ Οι πύργοι του Hanoi



- Οι πύργοι του Hanoi



```
void solve (int rings, int source,
           int target, int auxil) {
    if (rings == 0) return;
    solve(rings-1, source, auxil, target);
    WRITELN("Move from", source,
           "to", target);
    solve(rings-1, auxil, target, source);
}
```

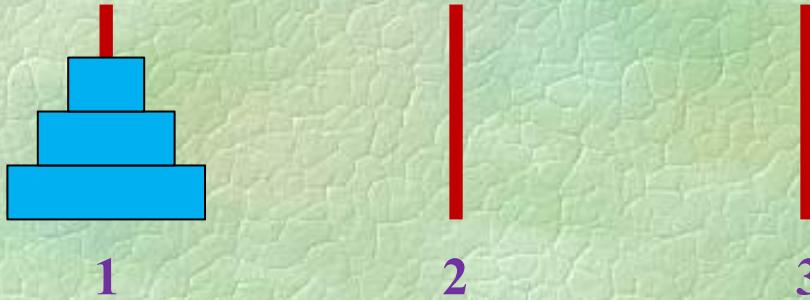
## □ Οι πύργοι του Hanoi

**PROGRAM** {

```
    WRITE("Rings: ") ;
    int rings = READ_INT();
    solve(rings, 1, 3, 2);
}
```

Rings: 3 ↵

```
Move from 1 to 3
Move from 1 to 2
Move from 3 to 2
Move from 1 to 3
Move from 2 to 1
Move from 2 to 3
Move from 1 to 3
```



□ Συνάρτηση παρόμοια με του Ackermann

$$z(i, j, 0) = j+1 \quad z(i, 0, 1) = i$$

$$z(i, 0, 2) = 0 \quad z(i, 0, n+3) = 1$$

$$z(i, j+1, n+1) = z(i, z(i, j, n+1), n) \quad , \forall i, j, n \in \mathbb{N}$$

```
int z (int i, int j, int n) {  
    if (n==0) return j+1;  
    else if (j==0)  
        if (n==1) return i;  
        else if (n==2) return 0;  
        else return 1;  
    else return z(i, z(i, j-1, n), n-1);  
}
```

# Αμοιβαία αναδρομή

```
bool is_odd (int n);      // function prototype
```

```
bool is_even (int n) {  
    if (n == 0) return true;  
    else         return is_odd(n-1);  
}
```

```
bool is_odd (int n) {  
    if (n == 0) return false;  
    else         return is_even(n-1);  
}
```

### □ Τύπος **REAL**

- προσεγγίσεις πραγματικών αριθμών
- **trunc**: ακέραιο μέρος (αποκοπή)
- **floor**: ακέραιος που δεν υπερβαίνει
- **round**: στρογγυλοποίηση

### □ Παράσταση κινητής υποδιαστολής

- mantissa και εκθέτης  $\pm m \cdot 2^x$   
όπου  $0.5 \leq m < 1$  και  $x \in \mathbf{Z}$  ή  $m = x = 0$
- το  $m$  είναι περιορισμένης ακρίβειας,  
π.χ. 8 σημαντικά ψηφία

### □ Αριθμητικά σφάλματα

$$1000000 + 0.000000001 = 1000000 \quad \text{γιατί};$$

### □ Αναπαράσταση των αριθμών

$$1000000 \approx 0.95367432 \cdot 2^{20}$$

$$0.000000001 \approx 0.53687091 \cdot 2^{-29}$$

$$\approx 0.00000000 \cdot 2^{20}$$

$$\text{άθροισμα} \approx 0.95367432 \cdot 2^{20}$$

# Εύρεση τετραγωνικής ρίζας (i)

- Χωρίς χρήση της συνάρτησης **sqrt**
- Μέθοδος Βαβυλωνίων
- Καταγράφεται για πρώτη φορά από τον Ήρωνα του Αλεξανδρέα
- Ειδική περίπτωση της γενικότερης μεθόδου του **Newton**, για την εύρεση της ρίζας οποιασδήποτε συνεχούς συναρτήσεως
$$f(y) = 0 \quad \text{εδώ: } f(y) = y^2 - x$$
για κάποιο δοθέν  $x$

# Εύρεση τετραγωνικής ρίζας (ii)

## □ Περιγραφή της μεθόδου

- Δίνεται ο αριθμός  $x > 0$
- Έστω προσέγγιση  $y$  της ρίζας, με  $y \leq \sqrt{x}$
- Έστω  $z = x / y$
- Το  $z$  είναι προσέγγιση της ρίζας, με  $\sqrt{x} \leq z$
- Για να βρω μια καλύτερη προσέγγιση, παίρνω το μέσο όρο των  $y$  και  $z$
- Επαναλαμβάνω όσες φορές θέλω

# Εύρεση τετραγωνικής ρίζας (iii)

- Χωρίς χρήση της συνάρτησης **sqrt**

$$y_0 = 1 \quad y_{i+1} = \frac{1}{2} \left( y_i + \frac{x}{y_i} \right)$$

- Παράδειγμα:  $\sqrt{37}$  (6.08276253)

$$y_0 = 1$$

$$y_4 = 6.143246$$

$$y_1 = 19$$

$$y_5 = 6.083060$$

$$y_2 = 10.473684$$

$$y_6 = 6.082763$$

$$y_3 = 7.003174$$

...

# Εύρεση τετραγωνικής ρίζας (iv)

```
REAL sqroot (REAL x) {  
    const REAL epsilon = 0.00001; // 1E-5  
    REAL older, newer = 1;  
  
    do {  
        older = newer;  
        newer = (older + x/older) / 2;  
    } while (! ( /* συνθήκη τερματισμού */ ));  
    return newer;  
}
```

# Εύρεση τετραγωνικής ρίζας (v)

## □ Εναλλακτικές συνθήκες τερματισμού

- Σταθερός αριθμός επαναλήψεων
  - ▶  $n == 20$
- Επιτυχής εύρεση ρίζας λάθος!
  - ▶  $\text{newer} * \text{newer} == x$
- Απόλυτη σύγκλιση
  - ▶  $\text{abs}(\text{newer} * \text{newer} - x) < \text{epsilon}$
- Σχετική σύγκλιση
  - ▶  $\frac{\text{abs}(\text{newer} * \text{newer} - x)}{\text{newer}} < \text{epsilon}$

# Εύρεση τετραγωνικής ρίζας (vi)

- Εναλλακτικές συνθήκες τερματισμού
  - Απόλυτη σύγκλιση κατά Cauchy
    - ▶  $\text{abs}(\text{newer} - \text{older}) < \text{epsilon}$
  - Σχετική σύγκλιση
    - ▶  $\text{abs}(\text{newer} - \text{older}) / \text{newer} < \text{epsilon}$

# Τριγωνομετρικές συναρτήσεις (i)

- Συνημίτονο με ανάπτυγμα Taylor

$$\cos(x) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}$$

- για τον όρο με δείκτη  $i+1$  έχουμε:

$$(-1)^{i+1} \frac{x^{2i+2}}{(2i+2)!} = - \left[ (-1)^i \frac{x^{2i}}{(2i)!} \right] \frac{x^2}{(2i+1)(2i+2)}$$

- οπότε αν  $n = 2i+1$  έχουμε:

$$newterm = -oldterm \frac{x^2}{n(n+1)}$$

# Τριγωνομετρικές συναρτήσεις (ii)

```
REAL mycos (REAL x) {  
    const REAL epsilon = 1E-5;  
    REAL sqx = x * x, term = 1, sum = 1;  
    int n = 0;  
  
    do {  
        n = n + 2;  
        term = -term * sqx / (n*(n-1));  
        sum = sum + term;  
    } while (abs(term/sum) >= epsilon);  
    return sum;  
}
```

# Αριθμητικά συστήματα

(i)

$$\begin{array}{l} 2024 \\ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\ 4 \cdot 10^0 = 4 \\ 2 \cdot 10^1 = 20 \\ 0 \cdot 10^2 = 0 \\ 2 \cdot 10^3 = 2000 \\ \Sigma = 2024 \end{array}$$

$x_{n-1} \dots x_1 x_0$  (b)

$$\begin{array}{l} 101010_{(2)} \\ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\ 0 \cdot 2^0 = 0 \\ 1 \cdot 2^1 = 2 \\ 0 \cdot 2^2 = 0 \\ 1 \cdot 2^3 = 8 \\ 1 \cdot 2^3 = 0 \\ 1 \cdot 2^3 = 32 \\ \Sigma = 42 \end{array}$$

$$x_{n-1} \cdot b^{n-1} + \dots + x_1 \cdot b^1 + x_0 \cdot b^0 = \sum_{k=0}^{k < n} x_k \cdot b^k$$

$x_{n-1} \dots x_1 x_0$  (*b*)

$$x_{n-1} \cdot b^{n-1} + \dots + x_1 \cdot b^1 + x_0 \cdot b^0 = \sum_{k=0}^{k < n} x_k \cdot b^k$$

- Βάση *b*: τα  $n$  ψηφία είναι μεταξύ 0 και  $b-1$
- Το πρώτο (πλέον σημαντικό) ψηφίο  $x_{n-1}$  είναι μη μηδενικό
- Εξαίρεση για το μηδέν:  $n = 1$  και  $x_0 = 0$

- Ενδιαφέροντα υπολογιστικά προβλήματα:
- Υπολογισμός αριθμού, δεδομένων των ψηφίων της αναπαράστασής του με βάση  $b$
- Υπολογισμός των ψηφίων της αναπαράστασης ενός αριθμού με βάση  $b$
- Μετατροπή αναπαράστασης ενός αριθμού από ένα σύστημα με βάση  $b_1$  σε ένα άλλο σύστημα με βάση  $b_2$

# Αριθμητικά συστήματα (iv)

## □ Αναπαράσταση αριθμού

```
const int b;           // η βάση
int n;                // το πλήθος των ψηφίων
int digits[42];       // τα ψηφία
```

## □ Υπολογισμός αριθμού

```
int result = 0, p = 1;
for (int k = 0; k < n; ++k) {
    // Αναλλοίωτη:  $p = b^k$ 
    result = result + digits[k] * p;
    p = p * b;
}
```

- Υπολογισμός ψηφίων αναπαράστασης
- 2024 στο σύστημα με βάση 7;

$$\begin{array}{r} 5 \Big| 7 \\ \hline 5 \quad 0 \end{array} \quad \begin{array}{r} 41 \Big| 7 \\ \hline 6 \quad 5 \end{array} \quad \begin{array}{r} 289 \Big| 7 \\ \hline 2 \quad 41 \end{array} \quad \begin{array}{r} 2024 \Big| 7 \\ \hline 1 \quad 289 \end{array}$$

$$5621_{(7)} = 5 \cdot 7^3 + 6 \cdot 7^2 + 2 \cdot 7^1 + 1 \cdot 7^0$$

## □ Υπολογισμός ψηφίων αναπαράστασης

```
int number;           // ο δοθείς αριθμός  
  
n = 0;  
  
while (number > 0) {  
    digits[n] = number % b;  
    n = n + 1;  
    number = number / b;  
}  
  
if (n == 0) { digits[0] = 0; n = 1; }
```

- Τύποι δεδομένων
  - Ακέραιοι αριθμοί  
`int`      `char`
  - Καθορισμός προσήμανσης  
`signed` `unsigned`
  - Καθορισμός μεγέθους  
`short`      `long`
  - Αριθμοί κινητής υποδιαστολής  
`float`      `double`

# Από το **PZhelp** στη C++ (ii)

**char , signed char , unsigned char**

**signed short int , unsigned short int**

**signed int , unsigned int**

**signed long int , unsigned long int**

**float**

**double** **(REAL)**

**long double**

Με κόκκινο χρώμα όσα μπορούν να παραλειφθούν.

### □ Πρόγραμμα και υποπρογράμματα

```
int main () {  
    ...  
    // προαιρετικό  
    return 0;  
}
```

```
void p (...)  
{ ... }
```

```
int f (...)  
{ ... }
```

```
PROGRAM {  
    ...  
}
```

```
void p (...)  
{ ... }
```

```
int f (...)  
{ ... }
```

## □ Ανάθεση

```
x += 42;
i %= n+1;
```

```
x = y = z = 0;
```

```
y = (x = 17) + 25;
```

```
i++; /* ή */
i--; /* ή */
```

```
i = 3; x = i++;
i = 3; x = ++i;
```

```
i = i++; // λάθος!
```

```
x = x + 42;
i = i % (n+1);
```

```
x = 0; y = 0; z = 0;
```

```
x = 17; y = x + 25;
```

```
i = i+1;
i = i-1;
```

```
i = 3; x = i; i = i+1;
i = 3; i = i+1; x = i;
```

# Από το **PZhelp** στη C++ (v)

## □ Βρόχος for

```
for (i=1; i<=10; i++)
    ...
for (i=8; i>=1; i--)
    ...
for (i=1; i<=10; i+=2)
    ...
```

// Στη C++ κανείς δε μας εμποδίζει να γράψουμε αυτό:  
// (πολλές κακές ιδέες μαζεμένες...)

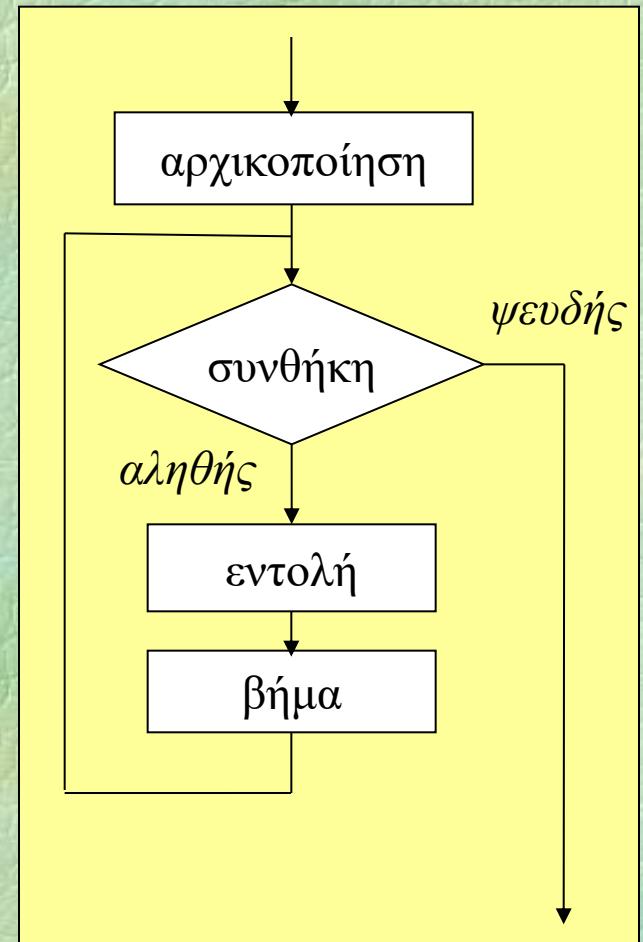
```
n=100; s=1;
for (i=1; i<=n; i+=s) {
    n-=i; s++; if (i+s>n) i=1;
}
```

- Βρόχος for

**for** (αρχικοποίηση ; συνθήκη ; βήμα)  
εντολή

```
s = 0;
// εσωτερική δήλωση της μεταβλητής i
for (int i=1; i <= 10; i++)
    s += i;

// προσέξτε τον τελεστή , (κόμμα)
int i, s;
for (i=1, s=0; i <= 10; i++)
    s += i;
```



# Τελεστής comma – Παράδειγμα

```
/* Comma as separator in the first line  
and as operator in the second line.  
Increases value of a by 2, then assigns  
value of resulting operation a+b into i.  
Results: a = 3, b = 2, c = 3, i = 5 */
```

```
int a = 1, b = 2, c = 3;  
int i = (a+=2, a+b);
```

## □ Έξοδος στην οθόνη στη C++

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

```
cout << "Hello\n";
/* ή */
cout << "Hello" << endl;
```

```
cout << i+1;
cout << i << " " << r;
cout << c;
```

```
WRITELN("Hello");
```

```
WRITE(i+1);
WRITE(i, r);
WRITE(c);
```

- Είσοδος από το πληκτρολόγιο στη C++

```
#include <iostream>
using namespace std;
...
cin >> i;
cin >> r;

cin >> c;
/* ή */
c = cin.get();

cin.ignore(
    numeric_limits<streamsizer>::max(),
    '\n'); // τρομακτικό λίγο, έτσι δεν είναι;
```

```
i = READ_INT();
r = READ_REAL();
c = getchar();
SKIP_LINE();
```

- Έξοδος στην οθόνη στη C++ (και στη C)

```
#include <cstdio>
```

```
...
```

```
printf("Hello\n");
printf("%d", i+1);
printf("%d %lf", i, r);
printf("%c", c);
```

```
printf("%5d", i);
printf("%5.3lf", r);
```

```
printf("%c %d %c %d\n",
      'a', 97, 97, 'a');
```

```
WRITELN("Hello");
WRITE(i+1);
WRITE(i, r);
WRITE(c);
```

```
WRITE(FORM(i,5));
WRITE(FORM(r,5,3));
```

```
a 97 a 97
```

# Από το **PZhelp** στη C++ (x)

- Είσοδος από το πληκτρολόγιο στη C++ (και στη C)

```
#include <cstdio>
```

```
...
```

```
scanf ("%d", &i);
```

```
scanf ("%lf", &r);
```

```
c = getchar();
```

```
/* ή */
```

```
scanf ("%c", &c);
```

```
while (getchar() != '\n');
```

```
i = READ_INT();
```

```
r = READ_REAL();
```

```
c = getchar();
```

```
SKIP_LINE();
```

- Δείκτης (pointer): η διεύθυνση μιας περιοχής της μνήμης όπου βρίσκεται μια μεταβλητή
- Παράδειγμα

```
int *p;
```

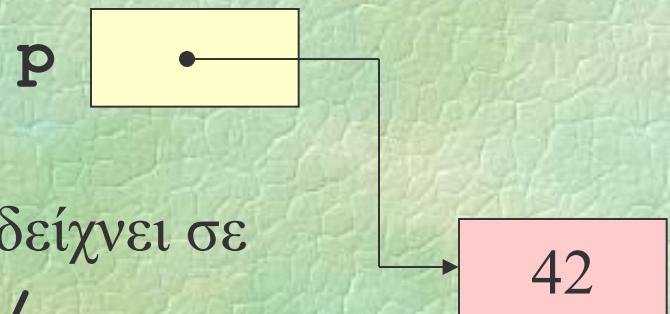
```
...
```

```
/* ο δείκτης p τοποθετείται να δείχνει σε  
κάποια ακέραια μεταβλητή */
```

```
...
```

```
*p = 42;
```

```
WRITELN(*p + 1);
```



## Δεικτοδότηση: &

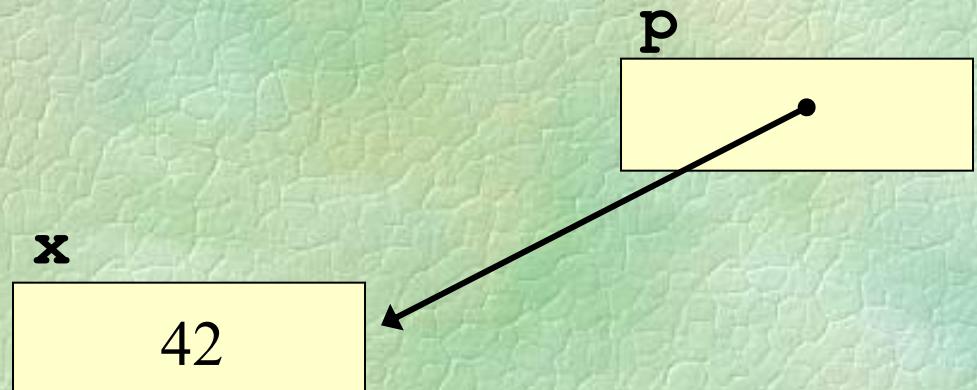
- η διεύθυνση μιας μεταβλητής

```
int x = 17, *p;  
p = &x;
```

## Αποδεικτοδότηση: \*

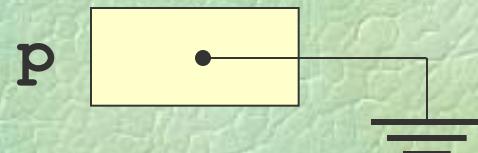
- το περιεχόμενο μιας διεύθυνσης

```
Writeln (*p);  
*p = 42;  
Writeln (x);
```



- Κενός δείκτης (**nullptr** ή **NULL**):  
ειδική τιμή δείκτη που «δε δείχνει πουθενά»
- Παράδειγμα

```
int *p;  
...  
p = nullptr;
```



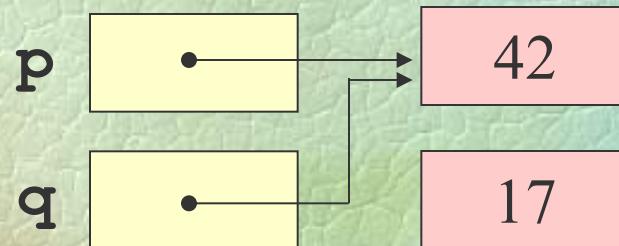
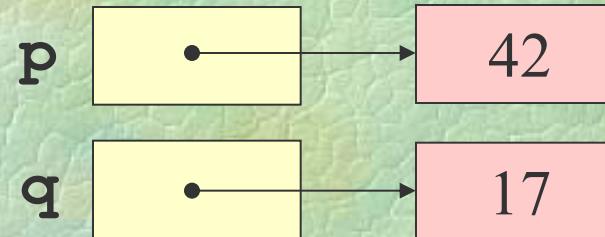
- Απαγορεύεται η προσπέλαση της μνήμης  
μέσω ενός κενού δείκτη

```
p = nullptr;  
WRITELN(*p); // λάθος!
```

# Δείκτες και ανάθεση

## □ Ανάθεση δεικτών

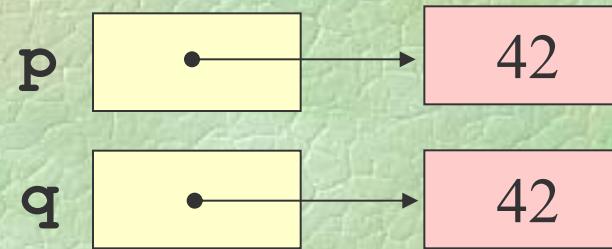
$q = p;$



πριν  
μετά

## □ Ανάθεση περιεχομένων

$*q = *p;$



μετά

# Παράδειγμα με δείκτες

```
PROGRAM { // pointers
    int x = 42, y = 17;
    int *p, *q;

    p = &x; q = &y;

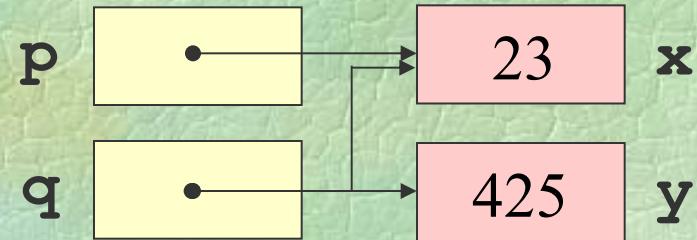
    *p = *p - *q;

    *q = *p * y;

    q = p;

    (*q)++; *p -= 3;

    WRITELN(x, y);
}
```



23 425

# Πέρασμα παραμέτρων με τιμή

```
int gcd (int a, int b);  
  
void normalize (int p, int q) {  
    int g = gcd(p, q);  
    p /= g;  q /= g;  
}  
  
PROGRAM { // call_by_value  
    int x = READ_INT();  
    int y = READ_INT();  
    normalize(x, y);  
    WRITELN(x, y);  
}
```

# Πέρασμα παραμέτρων με αναφορά

```
int gcd (int a, int b);

void normalize (int &p, int &q) {
    int g = gcd(p, q);
    p /= g;  q /= g;
}

PROGRAM { // call_by_ref
    int x = READ_INT();
    int y = READ_INT();
    normalize(x, y);
    WRITELN(x, y);
}
```

# Δείκτες αντί περάσματος με αναφορά

```
int gcd (int a, int b);  
  
void normalize (int *p, int *q) {  
    int g = gcd(*p, *q);  
    *p /= g;  *q /= g;  
}  
  
int main () {  
    int x, y;  
    scanf ("%d %d", &x, &y);  
    normalize (&x, &y);  
    printf ("%d %d\n", x, y);  
    return 0;  
}
```

Η C δεν υποστηρίζει  
πέρασμα με αναφορά!

## Αριθμητική δεικτών

```
int a[3] = {7, 6, 42};
```

```
int *p;
```

```
p = &(a[0]);
```

```
p = &a;
```

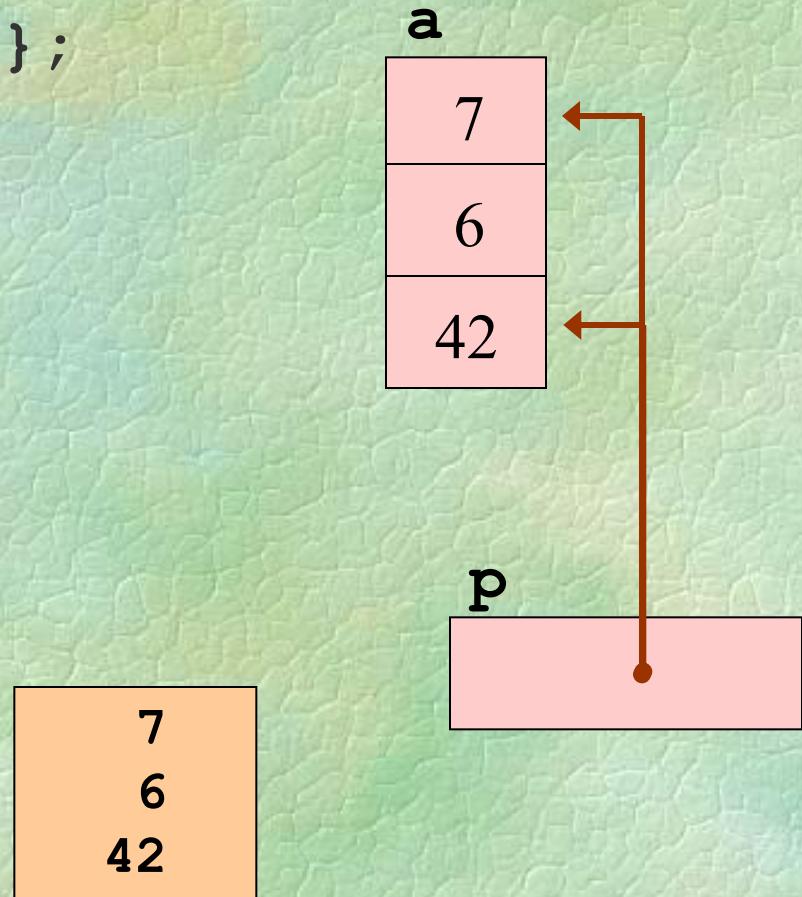
```
p = a;
```

```
WRITELN(*p);
```

```
WRITELN(* (p+1));
```

```
p = p+2;
```

```
WRITELN(*p);
```



## Ισοδυναμία πινάκων και δεικτών

- ▶ Ένας πίνακας είναι ένας δείκτης στο πρώτο στοιχείο.
- ▶  $a[i]$       ισοδύναμο με       $*(\alpha+i)$
- ▶ Οι πίνακες όμως είναι σταθεροί δείκτες, δηλαδή δεν μπορούν να αλλάξουν τιμή

```
int a[3] = {7, 6, 42};  
int *p = &a;  
p++; /* σωστό */  
a++; /* λάθος! */
```

## Συμβολοσειρές

```
char a[15] = "Hello world!", b[15];
// a[12] == '\0'

void strcpy (char *t, char *s) {
    while ((*t++ = *s++) != '\0');
}

int main () {
    strcpy(b, a);
    printf ("%s\n", b);
}
```

## Εκτύπωση συμβολοσειράς

```
void putchar (char c);

void puts (char *p) {
    while (*p != '\0') putchar(*p++);
}

int main () {
    ►     char s[] = "Hello world!\n";
    ►     puts(s);
    ► }
```

- Διάβασμα και επεξεργασία όλων των χαρακτήρων της εισόδου, π.χ. μέτρημα

```
int n = 0;
```

```
while (getchar() != EOF) n++;
printf("%d characters were read.\n", n);
```

- Η τιμή **EOF** σημαίνει το τέλος της εισόδου (**Ctrl-D** ή **Ctrl-Z** από το πληκτρολόγιο)

- Π.χ. αντιγραφή της εισόδου στην έξοδο

```
while (true) {  
    int c = getchar();  
    if (c == EOF) break;  
    putchar(c);  
}
```

- Η τιμή **EOF** πρέπει να ανατεθεί σε μεταβλητή **int**, όχι **char**! Ισοδύναμα:

```
int c;  
  
while ((c = getchar()) != EOF)  
    putchar(c);
```

# Επεξεργασία κειμένου (iii)

- Διάβασμα και επεξεργασία όλων των ακεραίων της εισόδου, π.χ. άθροιση

```
int i, sum = 0;  
while (true) {  
    if (scanf("%d", &i) != 1) break;  
    sum += i;  
}
```

- Η **scanf** επιστρέφει το πλήθος των στοιχείων που διαβάστηκαν. Ισοδύναμα:

```
int i, sum = 0;  
while (scanf("%d", &i) == 1) sum += i;
```

- Παράδειγμα 1: πρόγραμμα που
  - διαβάζει ένα κείμενο από την είσοδο
  - μετράει τον αριθμό των χαρακτήρων και τον αριθμό των γραμμών
  - υπολογίζει το μέσο όρο μήκους γραμμής
- Μετράμε τα '**\n**' και τους υπόλοιπους χαρακτήρες
- Ελέγχουμε για τέλος εισόδου (**EOF**)
- Για το μέσο όρο, κάνουμε διαίρεση!

## □ Παράδειγμα 1

```
int lines = 0, chars = 0;  
  
while (true) {  
    int c = getchar();  
    if (c == EOF) break;  
    if (c == '\n') lines++; else chars++;  
}  
  
printf("%d lines were read\n", lines);  
  
if (lines > 0)  
    printf("%0.3lf characters per  
line\n",  
           1.0 * chars / lines) μετατροπή τύπου
```

## □ Καλύτερα:

**(double) chars** (type cast)

- Παράδειγμα 2: πρόγραμμα που
  - διαβάζει ένα κείμενο από την είσοδο
  - μετράει τον αριθμό των χαρακτήρων, των λέξεων και των γραμμών
- Τι σημαίνει «λέξη»; Διαδοχικά γράμματα!
- Συνάρτηση για τον εντοπισμό γραμμάτων

```
bool isletter (char c) {  
    return c >= 'a' && c <= 'z'  
        || c >= 'A' && c <= 'Z';  
}
```

## □ Παράδειγμα 2

```
int c, lines = 0, chars = 0, words = 0;  
  
c = getchar();  
while (c != EOF)  
    if (isletter(c)) { words++;  
        do { chars++; c = getchar(); }  
        while (isletter(c));  
    }  
    else { chars++;  
        if (c == '\n') lines++;  
        c = getchar();  
    }
```

## □ Έχουμε διαβάσει ένα χαρακτήρα «μπροστά»!

- Παράδειγμα 3: πρόγραμμα που
  - διαβάζει ένα κείμενο από την είσοδο
  - μετράει τις συχνότητες εμφάνισης λέξεων με μήκος από 1 μέχρι 20 γράμματα
- Μέτρηση μήκους λέξης
- Μετρητές λέξεων ανά μήκος: πίνακας!
- Εδώ δε χρειάζεται να ασχοληθούμε με τις αλλαγές γραμμών!

## □ Παράδειγμα 3

```
int c, freq[21];  
  
for(int i=1; i<=20; i++) freq[i] = 0;  
  
c = getchar();  
while (c != EOF)  
    if (isletter(c)) {  
        int n = 0;  
        do { n++; c = getchar(); }  
        while (isletter(c));  
        if (n <= 20) freq[n]++;  
    }  
else c = getchar();
```

## □ Παράδειγμα 3 (συνέχεια)

```
for (int i=1; i<=20; i++)
    printf("%4d words of length %2d\n",
           freq[i], i);
```

## □ Μετατροπή κεφαλαίων γραμμάτων σε πεζά

```
char tolower (char ch) {
    if (ch >= 'A' && ch <= 'Z')
        return ch - 'A' + 'a';
    else
        return ch;
}
```

- Παράδειγμα 4: πρόγραμμα που
  - διαβάζει ένα κείμενο από την είσοδο
  - γράφει τους χαρακτήρες κάθε γραμμής αντίστροφα
- Αποθήκευση των χαρακτήρων κάθε γραμμής: πίνακας!
- Πρέπει να υποθέσουμε ένα μέγιστο μήκος γραμμής — θα έπρεπε να κάνουμε έλεγχο υπέρβασής του!

## □ Παράδειγμα 4

```
const int MAX = 80;
int c, line[MAX];

while ((c = getchar()) != EOF) {
    int n = 0;

    while (c != '\n') {
        line[n++] = c; c = getchar();
    }

    for (int i = n-1; i >= 0; i--)
        putchar(line[i]);
    putchar('\n');
}
```

## □ Εύρεση εμφάνισης λέξης-κλειδιού

...

```
// η λέξη-κλειδί έχει 3 χαρακτήρες  
for (int j = 0; j < 3; j++)  
    key[j] = getchar();
```

...

```
// έστω i το μήκος της γραμμής  
for (int k = 0; k+2 < i; k++)  
    if (line[k] == key[0] &&  
        line[k+1] == key[1] &&  
        line[k+2] == key[2])  
        WRITELN ("keyword found!");
```

# Συμβολοσειρές

(i)

- Πίνακες χαρακτήρων **char []**
  - Δείκτες σε χαρακτήρα **char \***
  - Τελειώνουν με το χαρακτήρα '**\0**'
  - Παράδειγμα

```
char name[30];  
  
printf("What's your name?\n");  
  
scanf("%s", name);  
  
printf("Hi %s, how are you?\n", name);
```

- Χρήσιμες συναρτήσεις βιβλιοθήκης

**#include <cstring>**

- Μέτρηση μήκους: **strlen**

```
printf("Your name has %d letters.\n",
       strlen(name));
```

- Λεξικογραφική σύγκριση: **strcmp**

```
if (strcmp(name, "John") == 0)
    printf("I knew you were John!\n");
```

- Quiz: **strcmp("ding", "dong") == ?**

□ Αντιγραφή:

**strcpy**

```
char a[10];  
strcpy(a, "ding");  
a[1] = 'o';  
printf("%s\n", a); // dong
```

□ Συνένωση:

**strcat**

```
char a[10] = "abc";  
strcat(a, "def");  
printf("%s\n", a); // abcdef
```

- Πρόβλημα: να αναδιαταχθούν τα στοιχεία ενός πίνακα ακεραίων σε αύξουσα σειρά
- Μια από τις σημαντικότερες εφαρμογές των ηλεκτρονικών υπολογιστών
- Βασική διαδικασία: εναλλαγή τιμών

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

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 5 | 7 | 2 | 1 | 8 | 3 | 6 | 4 |
|---|---|---|---|---|---|---|---|

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|

# Συγκριτικές Μέθοδοι Ταξινόμησης

- **Αντιμετάθεση** κάθε ζεύγους στοιχείων εκτός διάταξης (bubble sort).
- **Εισαγωγή** στοιχείου σε κατάλληλη θέση ταξινομημένου υποπίνακα (insertion sort).
- **Επιλογή** μεγαλύτερου στοιχείου και τοποθέτηση στο τέλος (selection sort, heapsort).
- **Συγχώνευση** ταξινομημένων πινάκων :  
Διαίρεση στη μέση, ταξινόμηση, συγχώνευση (mergesort).
- **Διαίρεση** σε μικρότερα και μεγαλύτερα από στοιχείο-διαχωρισμού και ταξινόμηση (quicksort).

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 5 | 7 | 2 | 1 | 8 | 3 | 6 | 4 |
|---|---|---|---|---|---|---|---|

- Μέθοδος της φυσαλίδας (bubble sort)

```
for (i = 0; i < n-1; i++)
    for (j = n-2; j >= i; j--)
        if (a[j] > a[j+1])
            swap(a[j], a[j+1]);
```

- Πλήθος συγκρίσεων

$$(n-1) + (n-2) + \dots + 2 + 1 = n(n-1)/2$$

της τάξης του  $n^2 \Rightarrow O(n^2)$

## □ Παράδειγμα εκτέλεσης ( $n = 7$ )

input: 12 4 9 8 6 7 5

---

12 4 9 8 6 5 7

12 4 9 8 5 6 7

12 4 9 5 8 6 7

12 4 5 9 8 6 7

12 4 5 9 8 6 7

$i = 0$       4 12 5 9 8 6 7

---

4 12 5 9 8 6 7

4 12 5 9 6 8 7

4 12 5 6 9 8 7

4 12 5 6 9 8 7

$i = 1$       4 5 12 6 9 8 7

---

4 5 12 6 9 7 8

4 5 12 6 7 9 8

4 5 12 6 7 9 8

$i = 2$       4 5 6 12 7 9 8

---

4 5 6 12 7 8 9

4 5 6 12 7 8 9

$i = 3$       4 5 6 7 12 8 9

---

4 5 6 7 12 8 9

$i = 4$       4 5 6 7 8 12 9

---

$i = 5$       4 5 6 7 8 9 12

---

## □ Βελτίωση με έλεγχο εναλλαγών

```
for (i = 0; i < n-1; i++) {  
    bool swaps = false;  
    for (j = n-2; j >= i; j--)  
        if (a[j] > a[j+1]) {  
            swaps = true;  
            swap(a[j], a[j+1]);  
        }  
    if (!swaps) break;  
}
```

## □ Στην καλύτερη περίπτωση απαιτούνται $O(n)$ συγκρίσεις, στη χειρότερη $O(n^2)$

## □ Ταξινόμηση με επιλογή (selection sort)

```
▶ for (int i=0; i<n-1; i++) {  
    int minpos=i;  
    ▶ for (int j=i+1; j<n; j++)  
        if (a[j] < a[minpos]) minpos = j;  
        swap(a[i], a[minpos]);  
}
```

## □ Πλήθος συγκρίσεων;

της τάξης του  $n^2 \Rightarrow O(n^2)$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 5 | 7 | 2 | 1 | 8 | 3 | 6 | 4 |
|---|---|---|---|---|---|---|---|

## □ Ταξινόμηση με εισαγωγή (insertion sort)

```

for (int i=1; i<n; i++) {
    int x = a[i], j = i;
    while (j > 0 && a[j-1] > x) {
        a[j] = a[j-1]; j = j-1;
    }
    a[j] = x;
}

```

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 5 | 7 | 2 | 1 | 8 | 3 | 6 | 4 |
|---|---|---|---|---|---|---|---|

$i = 1$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 5 | 7 | 8 | 3 | 6 | 4 |
|---|---|---|---|---|---|---|---|

$i = 5$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 5 | 7 | 2 | 1 | 8 | 3 | 6 | 4 |
|---|---|---|---|---|---|---|---|

$i = 2$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 5 | 7 | 8 | 6 | 4 |
|---|---|---|---|---|---|---|---|

$i = 6$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 1 | 8 | 3 | 6 | 4 |
|---|---|---|---|---|---|---|---|

$i = 3$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 5 | 6 | 7 | 8 | 4 |
|---|---|---|---|---|---|---|---|

$i = 7$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 5 | 7 | 8 | 3 | 6 | 4 |
|---|---|---|---|---|---|---|---|

$i = 4$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|

### □ Ταξινόμηση με εισαγωγή (insertion sort)

```
for (int i=1; i<n; i++) {  
    int x = a[i], j = i;  
    while (j > 0 && a[j-1] > x) {  
        a[j] = a[j-1]; j = j-1;  
    }  
    a[j] = x;  
}
```

### □ Πλήθος συγκρίσεων;

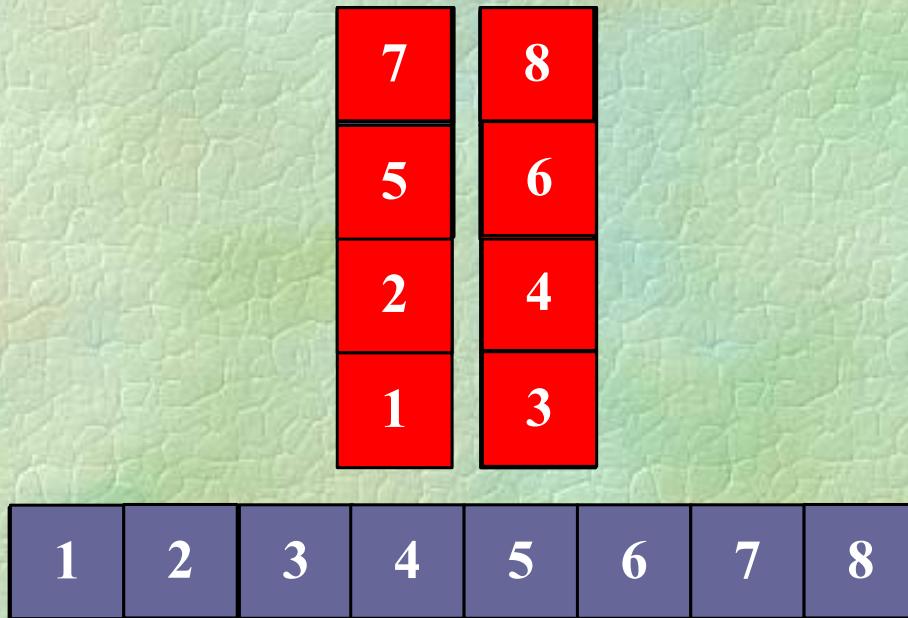
της τάξης του  $n^2 \Rightarrow O(n^2)$

- Ταξινόμηση με συγχώνευση (merge sort)
  - Διαιρώ την ακολουθία των αριθμών σε δύο μέρη
  - Με αναδρομικές κλήσεις, ταξινομώ τα δύο μέρη ανεξάρτητα
  - Συγχωνεύω τα δύο ταξινομημένα μέρη
- Στη χειρότερη περίπτωση απαιτούνται  $O(n \log n)$  συγκρίσεις

## □ Ταξινόμηση με συγχώνευση

```
void mergesort (int a[], int first,  
                int last) {  
  
    int mid;  
  
    if (first >= last) return;  
  
    mid = (first + last) / 2;  
    mergesort(a, first, mid);  
    mergesort(a, mid+1, last);  
    merge(a, first, mid, last);  
}
```

## □ Συγχώνευση



## □ Συγχώνευση

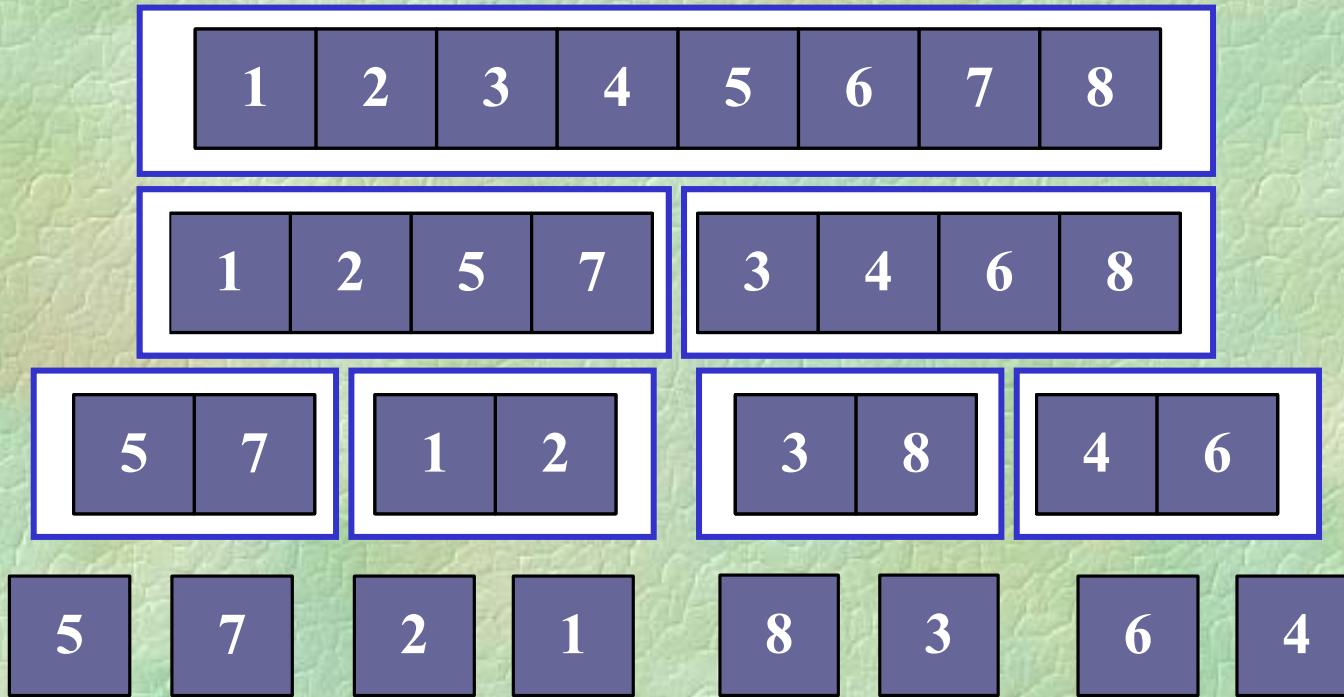
```
void merge (int a[], int first,
            int mid, int last) {
    int b[last-first+1];

    int i = first, j = mid+1, k = 0;
    while (i <= mid && j <= last)
        if (a[i] < a[j]) b[k++] = a[i++];
        else                  b[k++] = a[j++];

    while (i <= mid)      b[k++] = a[i++];
    while (j <= last)     b[k++] = a[j++];

    for(i=0; i<k; i++) a[first+i] = b[i];
}
```

## □ Διαδοχικές ταξινομήσεις και συγχωνεύσεις



## □ Ταξινόμηση με συγχώνευση (διαφορετικά)

```
void mergesort (int n, int *a) {  
    if (n <= 1) return;  
    int mid = n/2;  
    mergesort(mid, a);  
    mergesort(n-mid, a+mid);  
    merge(a, a+mid, a+n);  
}
```

## □ Συγχώνευση (διαφορετικά)

```
void merge (int *first, int *mid,
            int *last) {
    int b[last-first];
    int *i = first, *j = mid, *k = b;

    while (i < mid && j < last)
        if (*i < *j) *k++ = *i++;
        else           *k++ = *j++;

    while (i < mid)   *k++ = *i++;
    while (j < last) *k++ = *j++;

    i = first; j = b;
    while (j < k) *i++ = *j++;
}
```

## □ Ταξινόμηση με διαμέριση (quick sort)

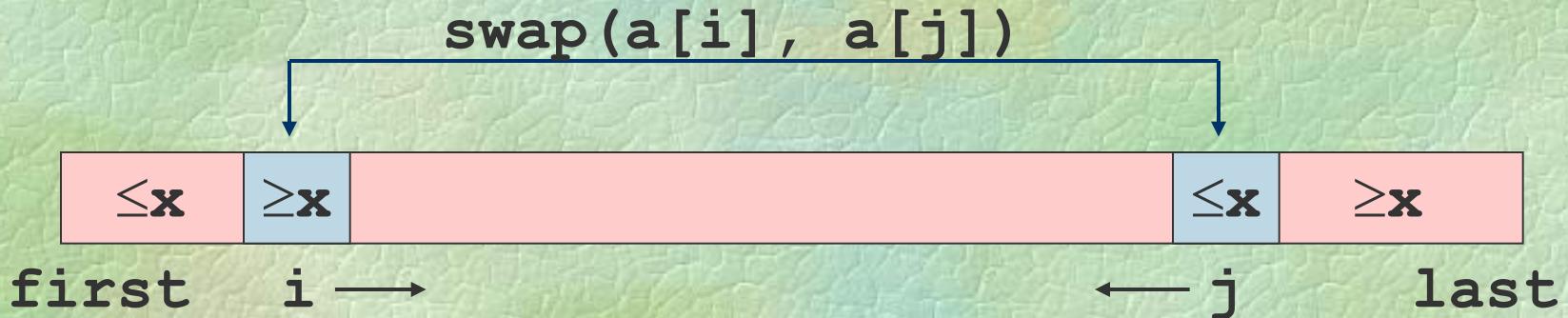
```
void quicksort (int a[], int first,  
                int last) {  
    int i;  
  
    if (first >= last) return;  
  
    i = partition(a, first, last);  
    quicksort(a, first, i);  
    quicksort(a, i+1, last);  
}
```

## □ Διαμέριση (partition)

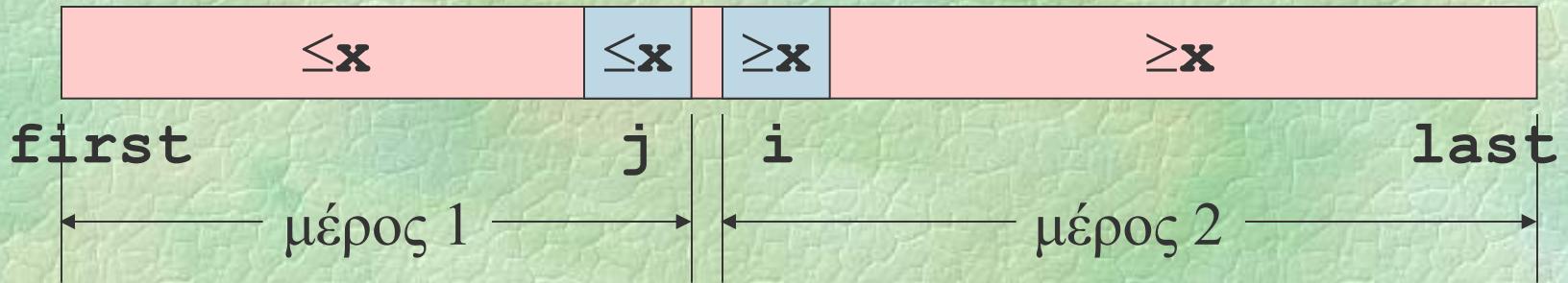
```
int partition (int a[], int first,
               int last) {
    // επιλογή ενός στοιχείου – συνήθως από random θέση
    int x = a[(first + last)/2];
    int i = first, j = last;

    while (true) {
        while (a[i] < x) i++;
        while (x < a[j]) j--;
        if (i >= j) break;
        swap(a[i], a[j]); i++; j--;
    }
    return j;
}
```

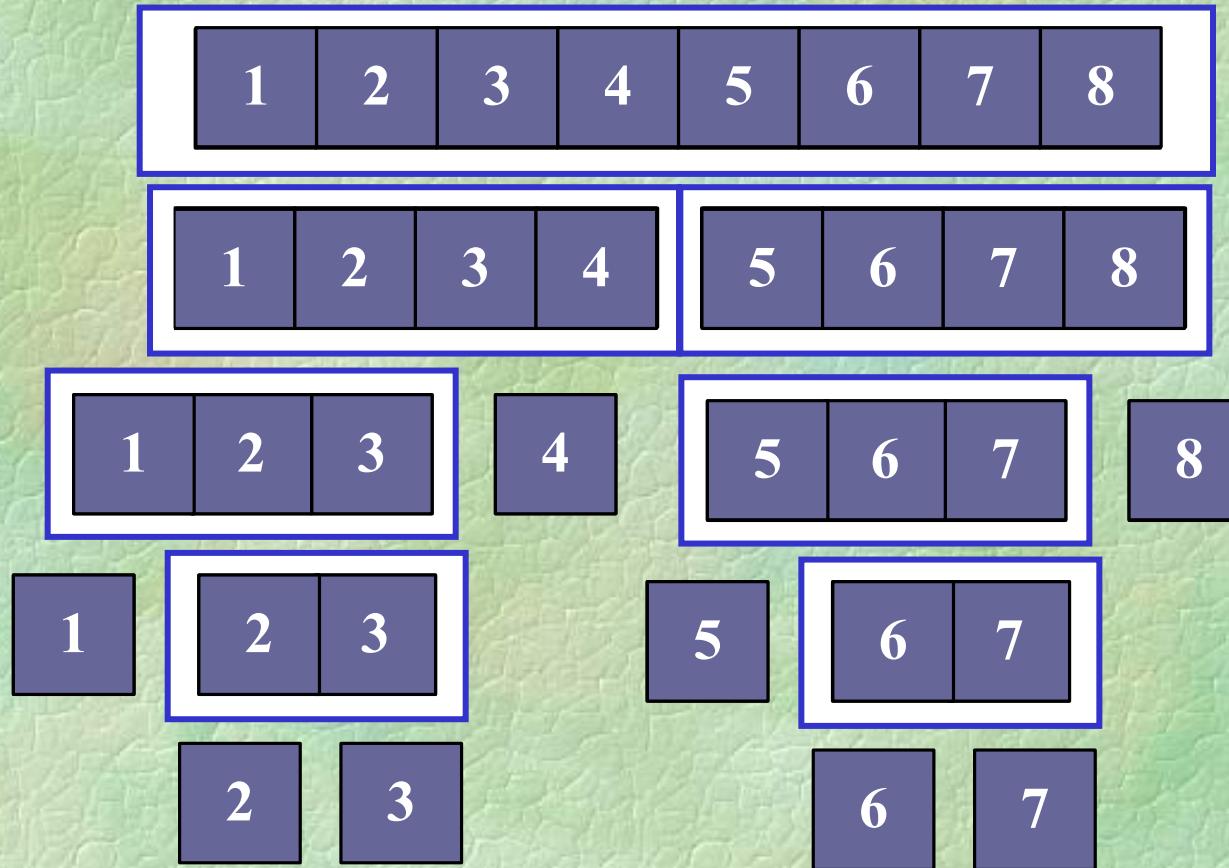
- Σε κάθε βήμα της διαμέρισης



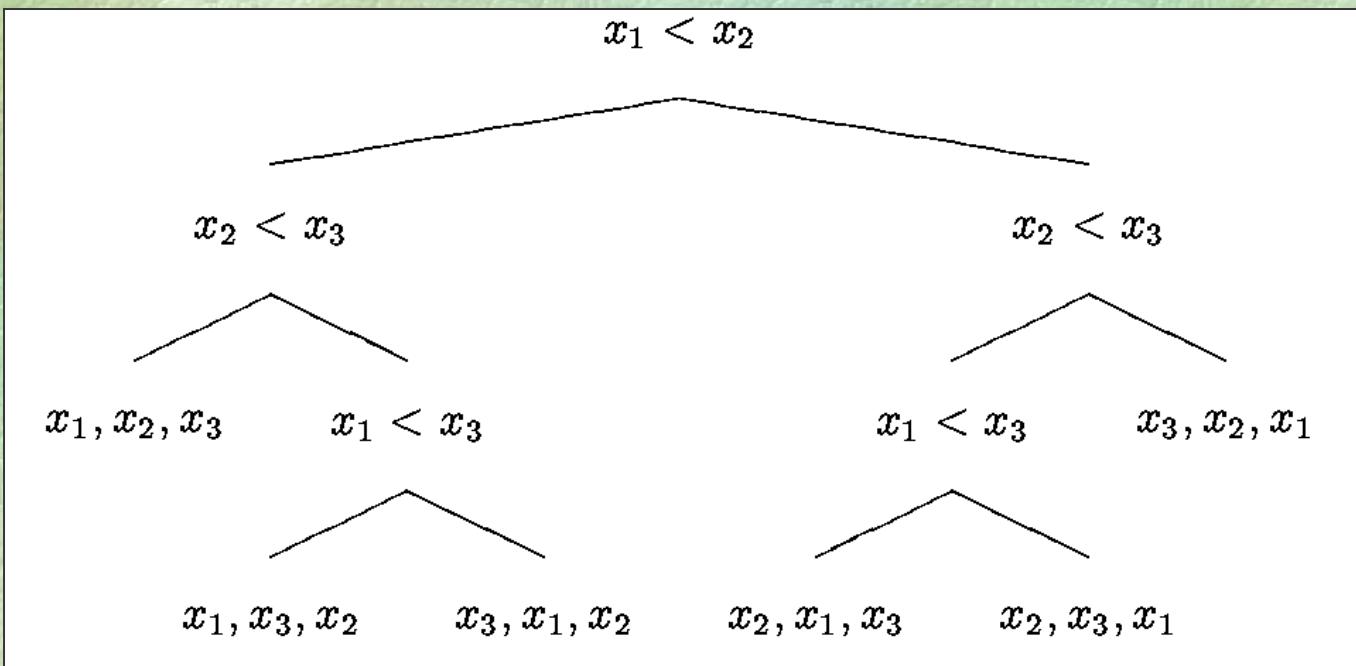
- Μετά τη διαμέριση



## □ Διαδοχικές διαμερίσεις και ταξινομήσεις



- Οποιοσδήποτε αλγόριθμος ταξινόμησης  $n$  αριθμών χρειάζεται τουλάχιστον  $\Omega(n \log n)$  συγκρίσεις



- Είδη ορθότητας
  - Συντακτική
  - Νοηματική
  - Σημασιολογική
- Σημασιολογική ορθότητα ελέγχεται:
  - με δοκιμές (testing)
  - με μαθηματική επαλήθευση

## □ Παράδειγμα: εύρεση γινομένου

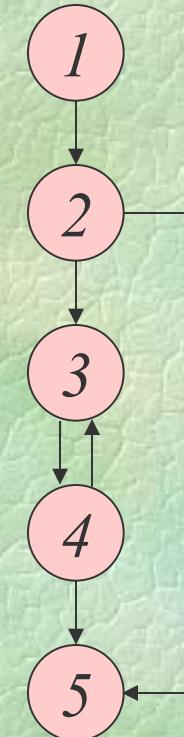
```
int mult (int x, int y) {  
    int i, z = 0;  
    for (i=1; i<=x; i++) z = z+y;  
    return z;  
}
```

## □ Ισχυρισμός:

- Η συνάρτηση υπολογίζει το γινόμενο δυο φυσικών αριθμών x και y

- Εντοπισμός σημείων όπου θα γραφούν βεβαιώσεις

```
int mult (int x, int y) {
    int i, /*1*/ z = 0; /*2*/
    for (i=1; i<=x; i++)
        /*3*/ z = z+y /*4*/ ;
    /*5*/ return z;
}
```



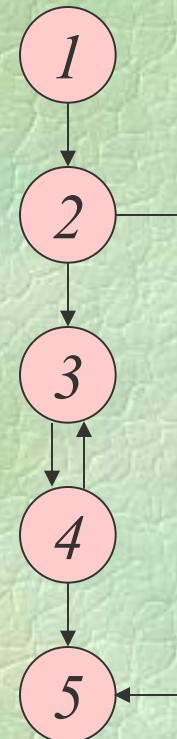
- Καταγραφή όλων των δυνατών τρόπων ροής ελέγχου

## □ Βεβαιώσεις

```

/*1 - Βεβαίωση εισόδου: x ≥ 0, y ≥ 0 */
z = 0;
/*2 : x ≥ 0, y ≥ 0, z = 0 */
for (i=1; i<=x; i++)
    /*3 - Αναλλοίωτη βρόχου:
       x ≥ 0, y ≥ 0, i ≤ x, z = y * (i-1) */
    z = z+y
    /*4 : x ≥ 0, y ≥ 0, z = y * i */
    ;
/*5 - Βεβαίωση εξόδου: x ≥ 0, y ≥ 0, z = y * x */
return z;

```



- Επαλήθευση: για κάθε δυνατό τρόπο ροής  
 $1 \rightarrow 2, 2 \rightarrow 3, 2 \rightarrow 5, 3 \rightarrow 4, 4 \rightarrow 3, 4 \rightarrow 5$

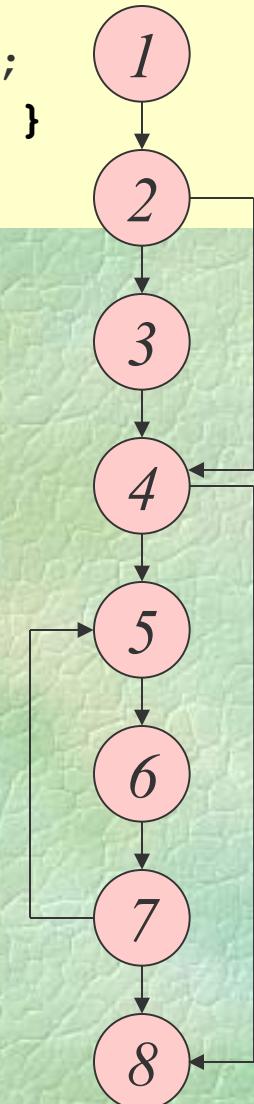
- Παράδειγμα: υπολογισμός δύναμης με επαναλαμβανόμενο τετραγωνισμό (Gauss)

```
REAL power (REAL y, int j) {  
    /*1*/ REAL x=y, z; int i=j; /*2*/  
    if (i<0) { /*3*/ x=1/x; i=abs(i); }  
    /*4*/ z=1;  
    while (i>0) {  
        /*5*/ if (i%2 != 0) z=z*x;  
        /*6*/ x=x*x; i=i/2; /*7*/  
    }  
    /*8*/ return z;  
}
```

# Ορθότητα

- Ροή ελέγχου
- Βεβαιώσεις

```
REAL power (REAL y, int j) {  
    /*1*/ REAL x=y, z; int i=j; /*2*/  
    if (i<0) { /*3*/ x=1/x; i=abs(i); }  
    /*4*/ z=1;  
    while (i>0) {  
        /*5*/ if (i%2 != 0) z=z*x;  
        /*6*/ x=x*x; i=i/2; /*7*/ }  
    /*8*/ return z; }
```



/\*1 – Βεβαίωση εισόδου: REAL y, int j \*/

/\*2 : x = y, i = j \*/

/\*3 : i < 0 \*/

/\*4 : i ≥ 0, y<sup>j</sup> = x<sup>i</sup> \*/

/\*5 – Αναλλοίωτη βρόχου: i ≥ 0, y<sup>j</sup> = z \* x<sup>i</sup> \*/

/\*6 : i ≥ 0, y<sup>j</sup> = z \* x<sup>i</sup> αν i άρτιος,  
y<sup>j</sup> = z \* x<sup>i-1</sup> αν i περιττός \*/

/\*7 : y<sup>j</sup> = z \* x<sup>i</sup> \*/

/\*8 – Βεβαίωση εξόδου: y<sup>j</sup> = z \*/

## □ Μερική ορθότητα (partial correctness)

- αν το πρόγραμμα σταματήσει,  
τότε το αποτέλεσμα θα είναι ορθό

## □ Ολική ορθότητα (total correctness)

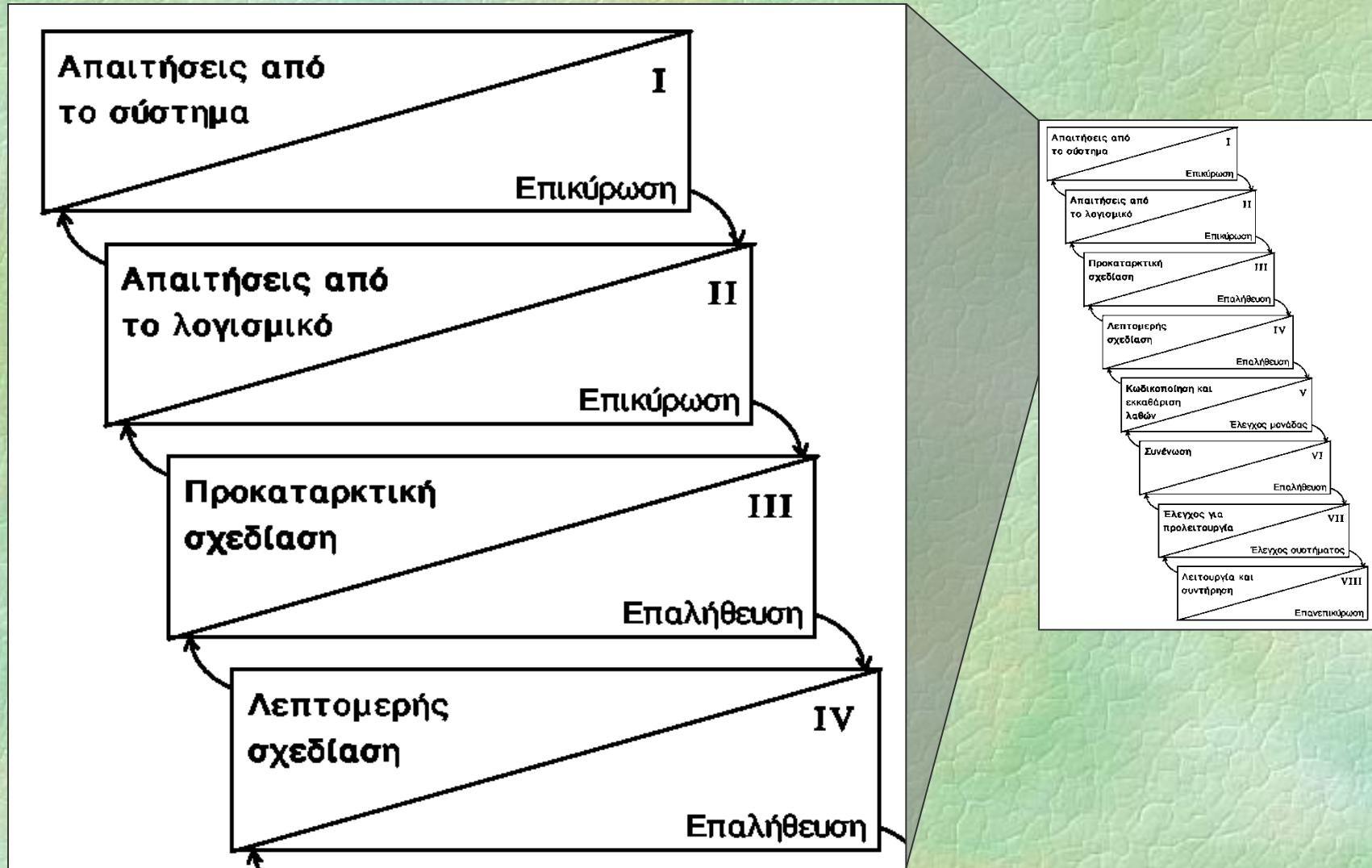
- το πρόγραμμα θα σταματήσει  
και το αποτέλεσμα θα είναι ορθό

# Τεχνολογία λογισμικού

- Software engineering
- Ανάπτυξη λογισμικού που να εξασφαλίζει:
  - παράδοση μέσα σε προδιαγεγραμμένα χρονικά όρια
  - κόστος μέσα σε προδιαγεγραμμένα όρια
  - καλή ποιότητα
  - αξιοπιστία
  - δυνατή και όχι δαπανηρή συντήρηση
- Μοντέλα κύκλου ζωής λογισμικού

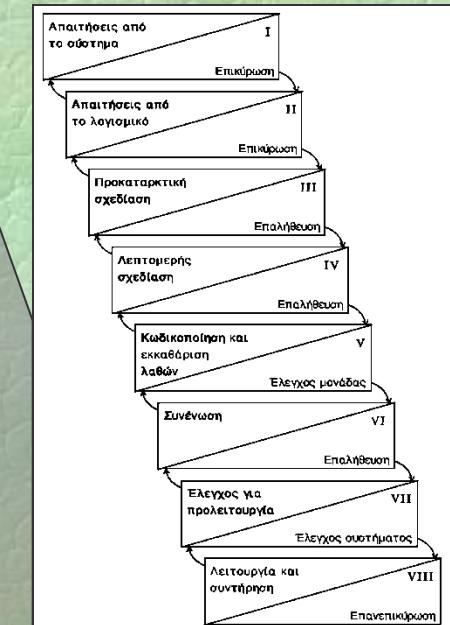
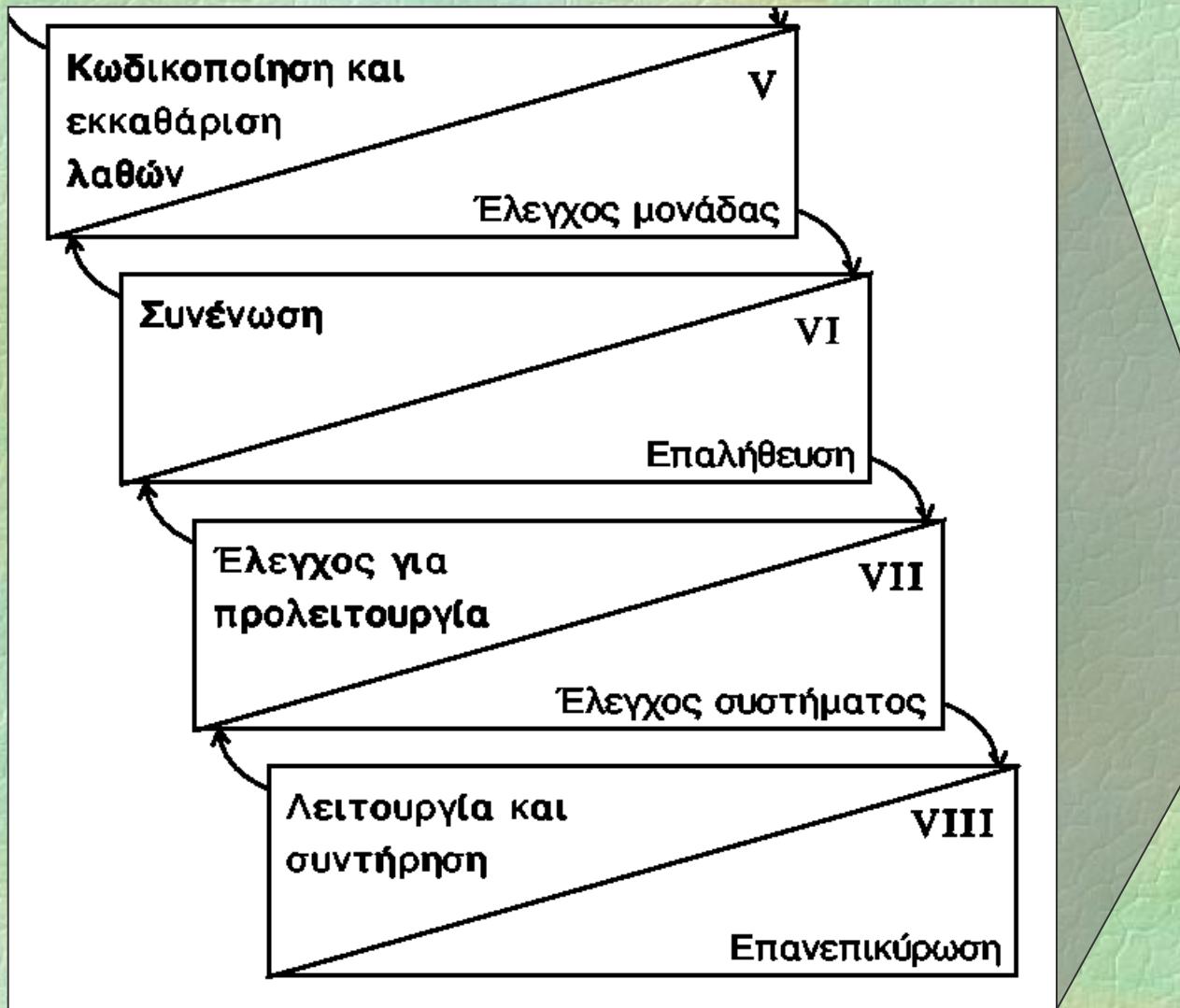
# Μοντέλο του καταρράκτη

(i)



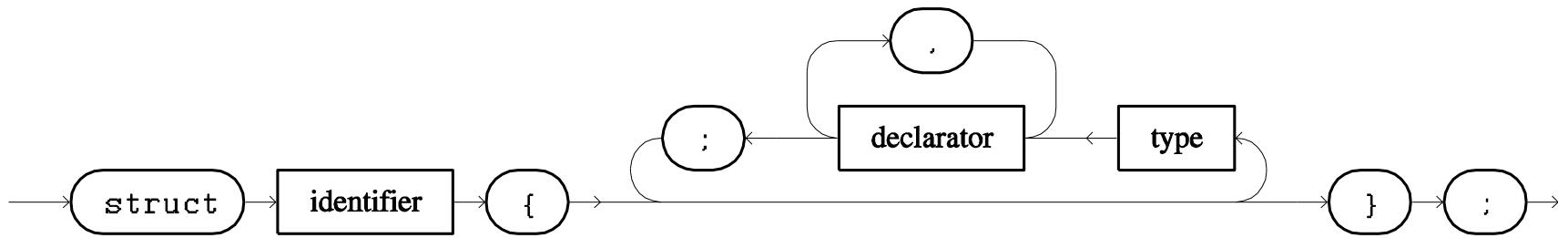
# Μοντέλο του καταρράκτη

(ii)



- Δομή (struct): δομημένη μεταβλητή που αποτελείται από πλήθος επιμέρους μεταβλητών πιθανώς διαφορετικών τύπων
- Οι επιμέρους μεταβλητές λέγονται **πεδία** και φέρουν ξεχωριστά ονόματα
- Σύνταξη

struct\_def



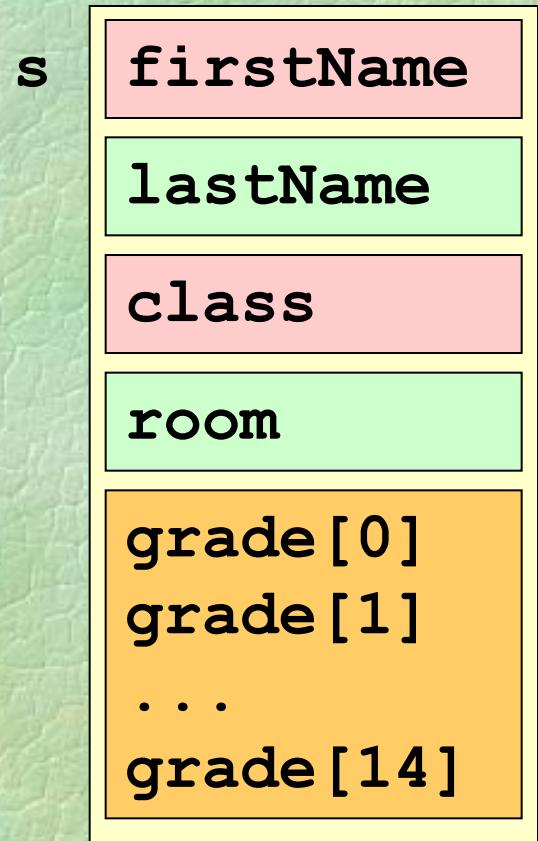
## □ Παράδειγμα

```
struct student {  
    char firstName[20];  
    char lastName[30];  
    int class, room;  
    int grade[15];  
};
```

```
student s;  
...
```

```
s.class = 3;
```

```
Writeln(s.firstName, s.lastName);
```



## □ Παράδειγμα: μέσος όρος βαθμολογίας

```
double average (student s) {  
    double sum = 0.0;  
  
    for (int i=0; i<15; i++)  
        sum += s.grade[i];  
    return sum / 15;  
}
```

```
struct date {  
    int day, month, year;  
};  
  
struct student {  
    ...  
    date birthDate;  
    ...  
};  
...  
  
WRITELN(s.birthDate.day, "/",  
           s.birthDate.month, "/",  
           s.birthDate.year);
```

Τι είναι οι μιγαδικοί αριθμοί;

say  
what!

Wikipedia: Στα μαθηματικά, οι μιγαδικοί αριθμοί είναι μία επέκταση του συνόλου των πραγματικών αριθμών με την προσθήκη του στοιχείου  $i$ , που λέγεται φανταστική μονάδα, και έχει την ιδιότητα:

$$i^2 = -1$$

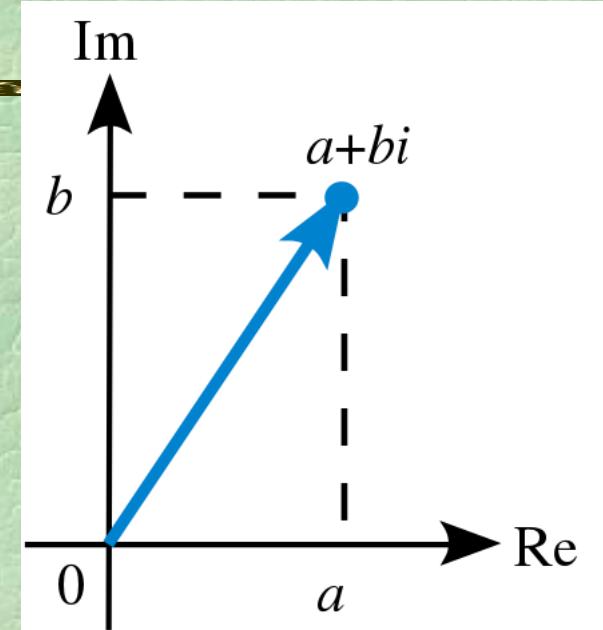
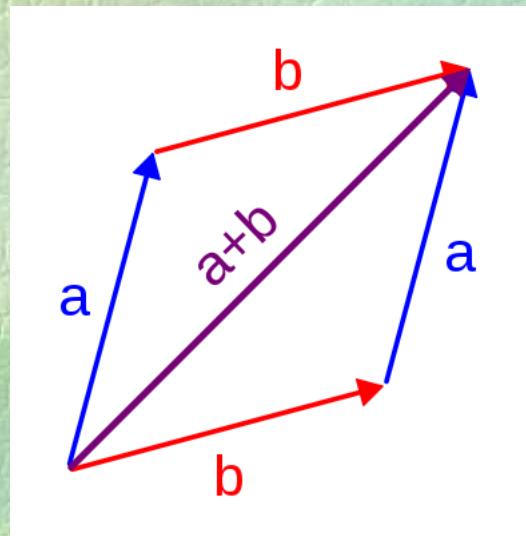
Κάθε μιγαδικός αριθμός μπορεί να γραφτεί στη μορφή  $a + bi$ , όπου  $a, b \in \mathbb{R}$

# Μιγαδικοί αριθμοί

a: πραγματικό μέρος

b: φανταστικό μέρος

Αριθμητικές πράξεις, π.χ.  
πρόσθεση



Για περισσότερα, ρωτήστε το μαθηματικό σας!

# Μιγαδικοί αριθμοί

```
struct complex { double re, im; };

complex cMult(complex x, complex y) {
    complex result;
    result.re = x.re * y.re - x.im * y.im;
    result.im = x.re * y.im + x.im * y.re;
    return result;
}

double cNorm(complex x) {
    return sqrt(x.re * x.re + x.im * x.im);
}
```

# Ενώσεις

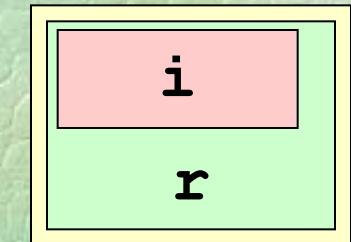
- Ένωση (union): όπως η δομή αλλά μόνο ένα από τα πεδία χρησιμοποιείται κάθε στιγμή!
- Παράδειγμα

```
union number { int i; double r; };

number n;
n.r = 1.2;
printf("%lf\n", n.r);

n.i = 42;
printf("%d\n", n.i);

printf("%lf\n", n.r); /* λάθος! */
```



## □ Αρχείο (file): αποτελείται από

- μεταβλητό πλήθος στοιχείων
- αποθηκευμένων το ένα μετά το άλλο
- συνήθως στην περιφερειακή μνήμη  
(π.χ. στο δίσκο)
- εν γένει περιέχει δυαδικά δεδομένα (binary)
- ειδική περίπτωση: αρχείο κειμένου

## □ Παράδειγμα

```
#include <cstdio>  
FILE *f;
```

### □ Άνοιγμα και κλείσιμο αρχείου

**fopen      fclose**

### □ Διάβασμα και γράψιμο

|                |               |                 |
|----------------|---------------|-----------------|
| <b>fputc</b>   | <b>fgetc</b>  | χαρακτήρες      |
| <b>fputs</b>   | <b>fgets</b>  | συμβολοσειρές   |
| <b>fprintf</b> | <b>fscanf</b> | οτιδήποτε       |
| <b>fwrite</b>  | <b>fread</b>  | ακολουθίες byte |

### □ Έλεγχος τέλους αρχείου

**feof**

# Αρχεία κειμένου στη C++ (i)

- Παράδειγμα: μέτρηση αριθμού γραμμών και χαρακτήρων πολλών αρχείων που ονομάζονται στη γραμμή εντολών

```
#include <cstdio>

int main (int argc, char *argv[]) {

    for (int i=1; i<argc; i++)
        // επεξεργασία του αρχείου argv[i]
}
```

# Αρχεία κειμένου στη C++ (ii)

## □ Παράδειγμα (συνέχεια)

```
// επεξεργασία των αρχείου argv[i]

FILE *f;
int chars = 0, lines = 0, c;

if ((f = fopen(argv[i], "rt")) ==
    nullptr)
    return 1;

while ((c = fgetc(f)) != EOF) {
    chars++;
    if (c == '\n') lines++;
}
fclose(f);
printf("%d chars, %d lines, %s\n",
       chars, lines, argv[i]);
```

### □ Παράδειγμα: αντιγραφή δυαδικών αρχείων

```
> #include <cstdio>  
  
> int main (int argc, char * argv[]) {  
>     FILE *fin, *fout;  
  
>     fin = fopen(argv[1], "rb");  
>     if (fin == nullptr) return 1;  
  
>     fout = fopen(argv[2], "wb");  
>     if (fout == nullptr) return 2;
```

### □ Παράδειγμα (συνέχεια)

```
>     while (!feof(fin)) {  
>         unsigned char buffer[1000];  
>         unsigned int count;  
>  
>         count = fread(buffer, 1, 1000,  
fin);  
>         fwrite(buffer, 1, count, fout);  
>     }  
>     fclose(fin);  
>     fclose(fout);  
> }
```

- Κόστος της εκτέλεσης ενός αλγορίθμου που επιλύει κάποιο πρόβλημα, συναρτήσει του μεγέθους του προβλήματος
  - χρόνος: αριθμός υπολογιστικών βημάτων
  - χώρος: απαιτούμενο μέγεθος μνήμης
- Συναρτήσεις πολυπλοκότητας
  - θετικές και αύξουσες
  - π.χ.  $f(n) = n(n-1)/2$

# Πολυπλοκότητα

## (ii)

□ Άνω φράγμα:  $O$

$$O(f) = \{ g \mid \exists c. \exists n_0. \forall n > n_0. g(n) < c f(n) \}$$

□ Κάτω φράγμα:  $\Omega$

$$\Omega(f) = \{ g \mid \exists c. \exists n_0. \forall n > n_0. g(n) > c f(n) \}$$

□ Τάξη μεγέθους:  $\Theta$

$$\Theta(f) = \{ g \mid \exists c_1, c_2. \exists n_0. \forall n > n_0. c_1 < g(n) / f(n) < c_2 \}$$

□ Γράφουμε  $g = O(f)$  αντί  $g \in O(f)$

□ π.χ.  $5n^2 + 4n - 2n \log n + 7 = \Theta(n^2)$

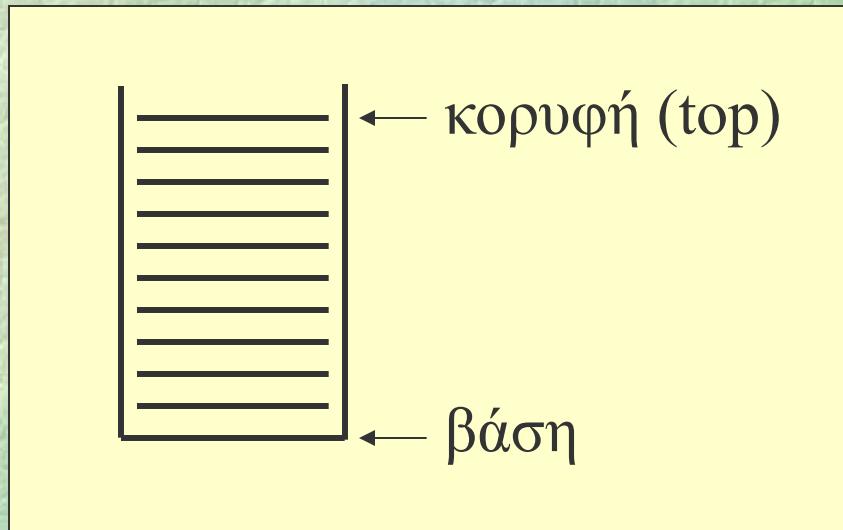
$O(1) < O(a(n)) < O(\log^* n) < O(\log n) < O(\sqrt{n})$   
 $< O(n) < O(n \log n)$   
 $< O(n^2) < O(n^2 \log^5 n)$   
 $< O(n^3) < \dots < \text{Poly}$   
 $< O(2^n) < O(n!) < O(n^n)$   
 $< O(2^{\wedge\wedge} n) < \dots < O(A(n))$

$\text{Poly} = n^{O(1)}$

$2^{\wedge\wedge} n$  η υπερεκθετική συνάρτηση:  $2^{2^{2^{\dots^2}}} (n \text{ φορές})$   
και  $\log^* n$  η αντίστροφή της

- ▶  $A(n)$  η συνάρτηση Ackermann και  $a(n)$  η αντίστροφή της

- Last In First Out (LIFO)  
ό,τι μπαίνει τελευταίο, βγαίνει πρώτο



### □ Αφηρημένος τύπος δεδομένων

- **stack**: υλοποιεί τη στοίβα (ακεραίων αριθμών)
- Ορίζονται οι απαραίτητες πράξεις:
  - (**stack**) δημιουργεί μια άδεια στοίβα
  - **empty** ελέγχει αν μια στοίβα είναι άδεια
  - **push** προσθήκη στοιχείου στην κορυφή
  - **pop** αφαίρεση στοιχείου από την κορυφή

- Ο τρόπος υλοποίησης των παραπάνω δεν ενδιαφέρει αυτούς που θα τα χρησιμοποιήσουν
  - Τέτοιοι τύποι λέγονται **αφηρημένοι** (ΑΤΔ)

- Αντικείμενα (objects) στη C++  
Υλοποίηση στοίβας με πίνακα

```
class stack {  
public:
```

```
    stack ();  
    bool empty ();  
    void push (int x);  
    int pop ();
```

κατασκευαστής (constructor)

```
private:
```

```
    const int size = 100;  
    int data[size];  
    int top;
```

μέθοδοι (methods)

πεδία (fields)

όπως στο struct

```
} ;
```

## □ Παράδειγμα χρήσης

```
PROGRAM { // stack_demo

stack s; // καλείται ο κατασκευαστής

for (int i = 1; i <= 10; i++)
    s.push(i);

while (!s.empty())
    WRITELN(s.pop());
}
```

- Κατασκευή άδειας στοίβας

```
stack::stack () {  
    top = 0;  
}
```

- Έλεγχος αν μια στοίβα είναι άδεια

```
bool stack::empty () {  
    return top == 0;  
}
```

## □ Προσθήκη στοιχείου

```
void stack::push (int x) {  
    data[top++] = x;  
}
```

## □ Αφαίρεση στοιχείου

```
int stack::pop () {  
    return data[--top];  
}
```

# Διαχείριση της μνήμης (i)

- **Στατικές μεταβλητές:** γενικές ή τοπικές
  - ο χώρος στη μνήμη όπου τοποθετούνται δεσμεύεται κάθε φορά που καλείται η ενότητα όπου δηλώνονται και αποδεσμεύεται στο τέλος της κλήσης
- **Δυναμικές μεταβλητές**
  - ο χώρος στη μνήμη όπου τοποθετούνται δεσμεύεται και αποδεσμεύεται δυναμικά, δηλαδή με φροντίδα του προγραμματιστή
  - η προσπέλαση σε δυναμικές μεταβλητές γίνεται με τη χρήση **δεικτών** (pointers)

- Με τη βοήθεια των δυναμικών μεταβλητών υλοποιούνται δυναμικοί τύποι δεδομένων
  - συνδεδεμένες λίστες,
  - δέντρα, γράφοι, κ.λπ.
- Πλεονεκτήματα των δυναμικών τύπων
  - μπορούν να περιέχουν **απεριόριστο πλήθος στοιχείων** (αν το επιτρέπει η διαθέσιμη μνήμη)
  - κάποιες πράξεις υλοποιούνται **αποδοτικότερα** (π.χ. προσθήκη και διαγραφή στοιχείων σε ενδιάμεση θέση)

# Δυναμική παραχώρηση μνήμης (i)

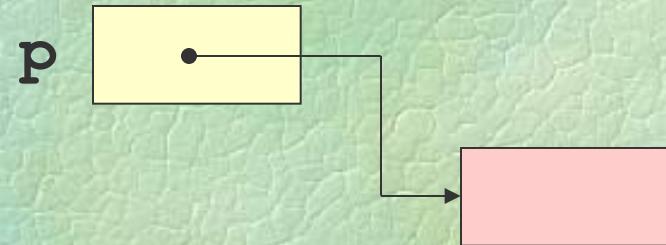
## □ Δέσμευση (C++)

- δημιουργία μιας νέας δυναμικής μεταβλητής

```
int *p;
```

```
...
```

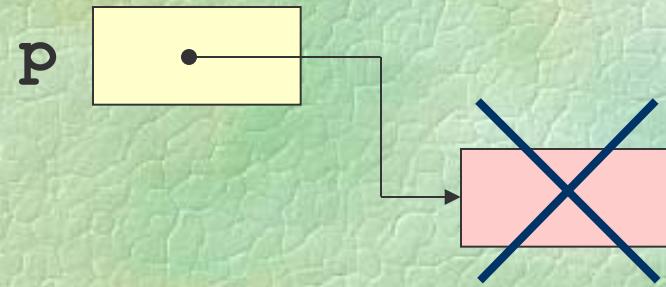
```
p = new int;
```



## □ Αποδέσμευση

- καταστροφή μιας δυναμικής μεταβλητής

```
delete p;
```



# Δυναμική παραχώρηση μνήμης (ii)

## □ Δέσμευση

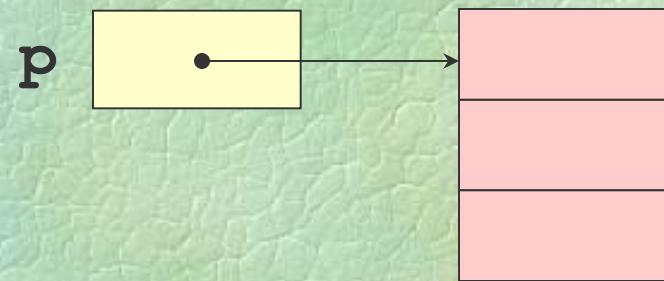
- δημιουργία πίνακα μεταβλητού μεγέθους

```
int *p, n;
```

```
...
```

```
n = 3;
```

```
p = new int[n];
```



// αν δεν υπάρχει αρκετή μνήμη, προκύπτει **εξαίρεση**

// σε αυτό το μάθημα δε θα μιλήσουμε για εξαιρέσεις στη C++

## □ Αποδέσμευση

```
delete [] p;
```

## Δυναμική παραχώρηση μνήμης (iii)

- Δέσμευση και αποδέσμευση σε C

```
#include <stdlib.h>

int *p;
...
```

```
p = new int[42];
...
delete [] p;
```

```
p = (int *) malloc(42 * sizeof(int));
if (p == NULL) {
    printf("Out of memory!\n");
    exit(1);
}
...
free(p);
```

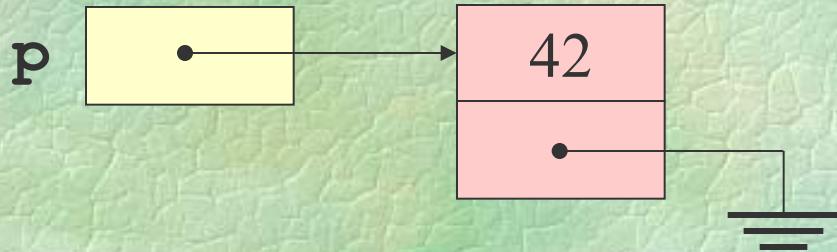
# Σύνθετες δυναμικές μεταβλητές (i)

## □ Παράδειγμα

```
struct thing {  
    int num;  
    int *ptr;  
};
```

```
thing *p;
```

```
p = new thing;  
p->num = 42;  
p->ptr = nullptr;
```



**p->something** ισοδύναμο με **(\*p).something**

# Σύνθετες δυναμικές μεταβλητές (ii)

## □ Παράδειγμα (συνέχεια)

`q = p;`

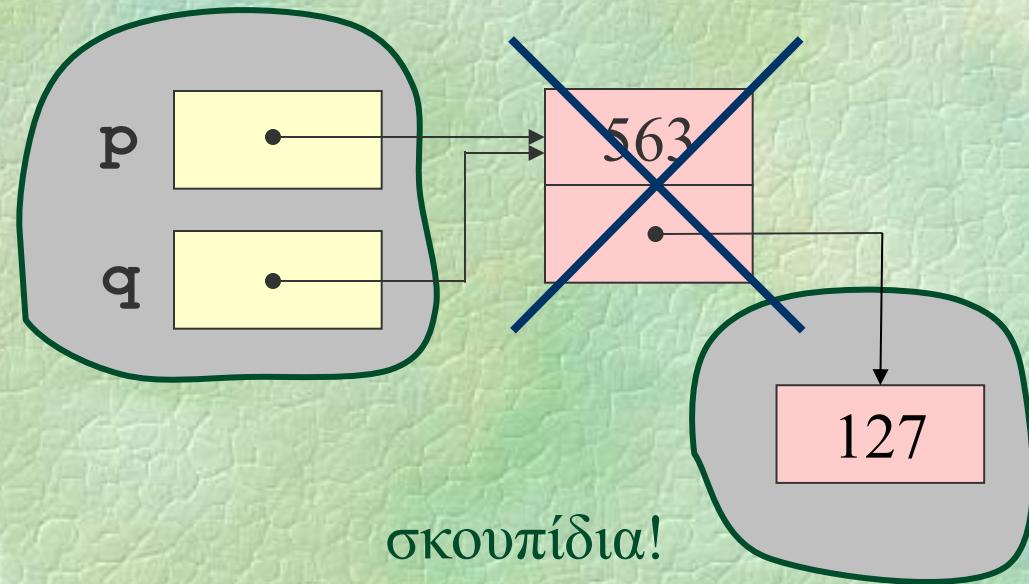
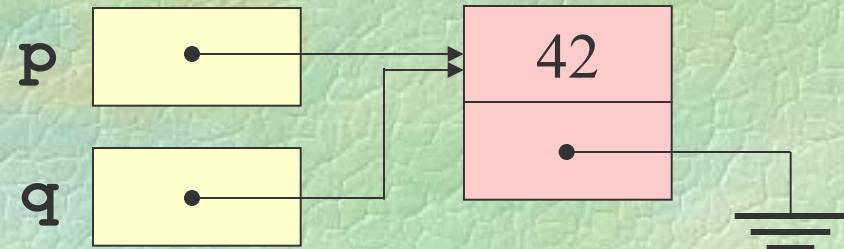
`q->num = 563;`

`q->ptr = new int;`

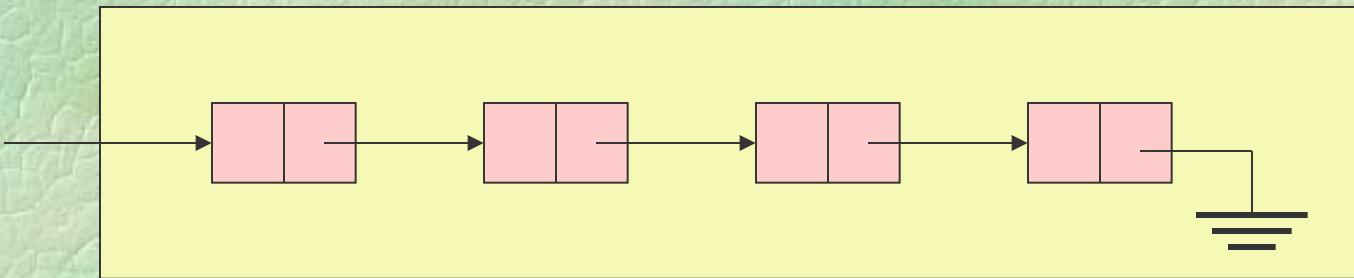
`*(q->ptr) = 127;`

`delete p;`

ξεκρέμαστοι δείκτες!

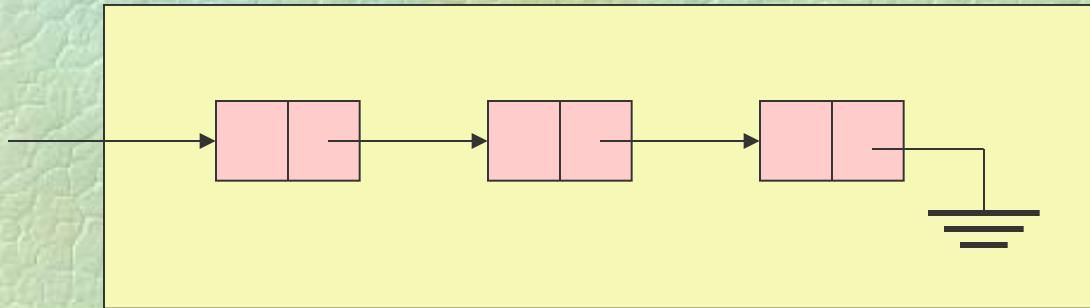


- Είναι γραμμικές διατάξεις
- Κάθε κόμβος περιέχει:
  - κάποια πληροφορία
  - ένα σύνδεσμο στον **επόμενο** κόμβο
- Ο τελευταίος κόμβος έχει **κενό σύνδεσμο**

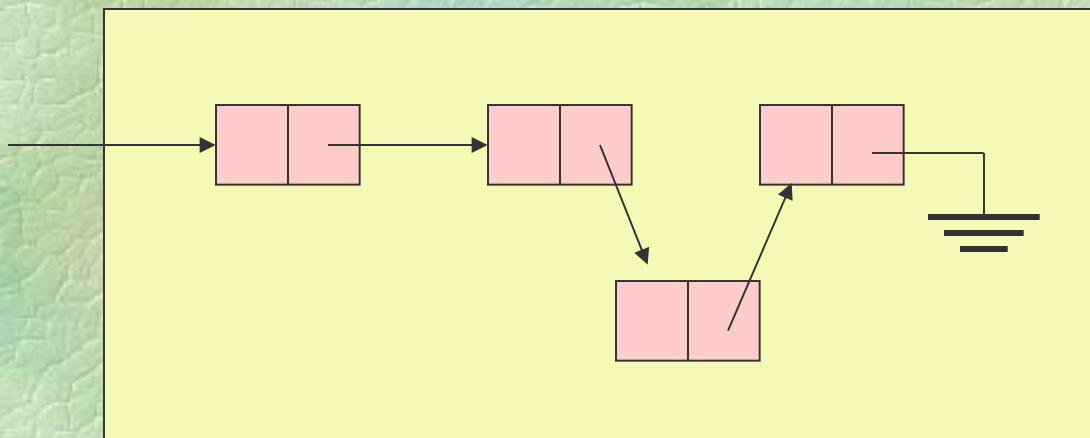


## □ Ευκολότερη προσθήκη στοιχείων

□ πριν

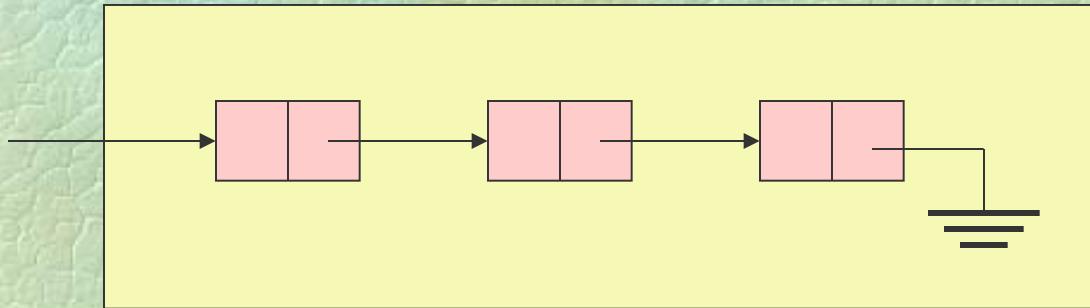


□ μετά

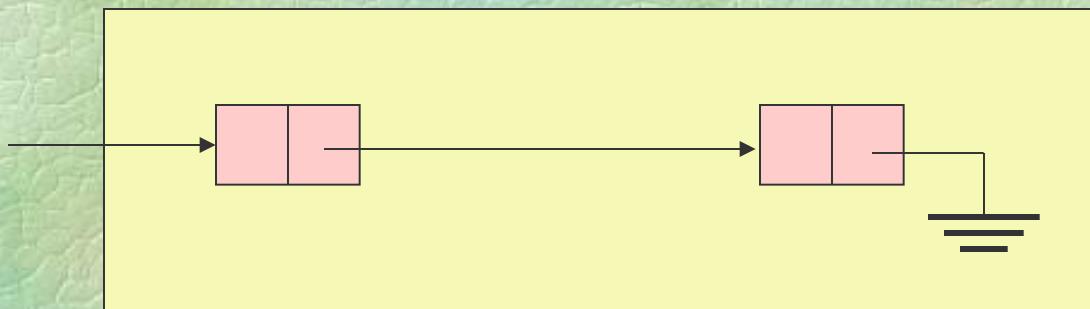


## □ Ευκολότερη διαγραφή στοιχείων

□ πριν



□ μετά



## □ Τύπος κόμβου συνδεδεμένης λίστας

```
struct node {  
    int info;  
    node *next;    ← αυτοαναφορά!  
};
```

## □ Μια συνδεδεμένη λίστα παριστάνεται συνήθως με ένα δείκτη στο πρώτο της στοιχείο

► **node \*head;**

## □ Παράδειγμα κατασκευής λίστας

```
node* readListReversed () {  
    node *head = nullptr, *n;  
    int data;  
  
    while (scanf("%d", &data) == 1) {  
        n = new node;  
        n->info = data;  
        n->next = head;  
        head = n;  
    }  
    return head;  
}
```

## □ Εκτύπωση λίστας

```
void print (node *p) {  
    while (p != nullptr) {  
        WRITELN (p->info) ;  
        p = p->next;  
    }  
}
```

## □ Ισοδύναμα (για να μη «χαθεί» η αρχή **p**):

```
for (node *q = p; q != nullptr;  
     q = q->next)  
    WRITELN (q->info) ;
```

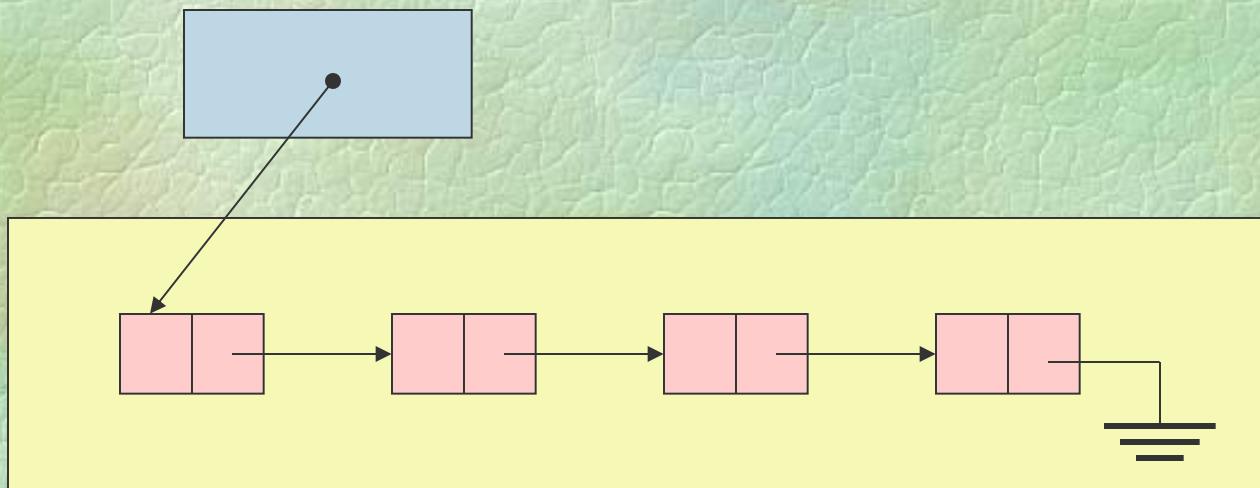
- Εκτύπωση λίστας με αναδρομή

```
void print (node *p) {  
    if (p != nullptr) { WRITELN(p->info) ;  
                        print(p->next) ;  
    }  
}
```

- Εκτύπωση λίστας αντίστροφα με αναδρομή

```
void printBack (node *p) {  
    if (p != nullptr) { printBack(p-  
        >next) ;  
                        WRITELN(p->info) ;  
    }  
}
```

- Υλοποίηση με απλά συνδεδεμένη λίστα



- Υλοποίηση με απλά συνδεδεμένη λίστα

```
class stack {  
public:  
    stack ();  
    bool empty ();  
    void push (int x);  
    int pop ();  
  
private:  
    struct node {  
        int info;  
        node *next;  
    };  
    node *head;  
};
```

ίδια όπως πριν!

- Κατασκευή άδειας στοίβας

```
stack::stack () {  
    head = nullptr;  
}
```

- Έλεγχος αν μια στοίβα είναι άδεια

```
bool stack::empty () {  
    return head == nullptr;  
}
```

## □ Προσθήκη στοιχείου

```
void stack::push (int x) {  
    node *p = new node;  
    p->info = x;  
    p->next = head;  
    head = p;  
}
```

## □ Αφαίρεση στοιχείου

```
int stack::pop () {  
    node *p = head;  
    int result = head->info;  
    head = head->next;  
    delete p;  
    return result;  
}
```

## □ Παράδειγμα χρήσης

```
PROGRAM { // stack_demo
```

```
stack s; // καλείται ο κατασκευαστής
```

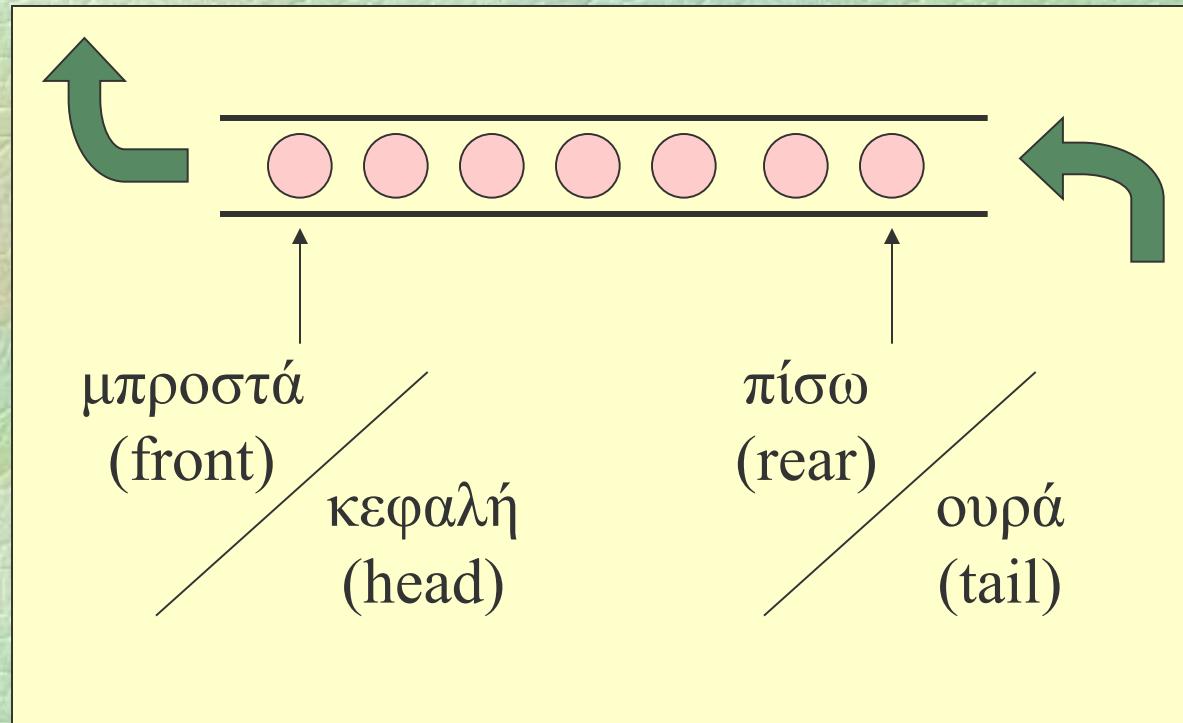
```
for (int i = 1; i <= 10; i++)
    s.push(i);
```

```
while (!s.empty())
    WRITELN(s.pop());
```

```
}
```

Δεν άλλαξε  
τίποτα!

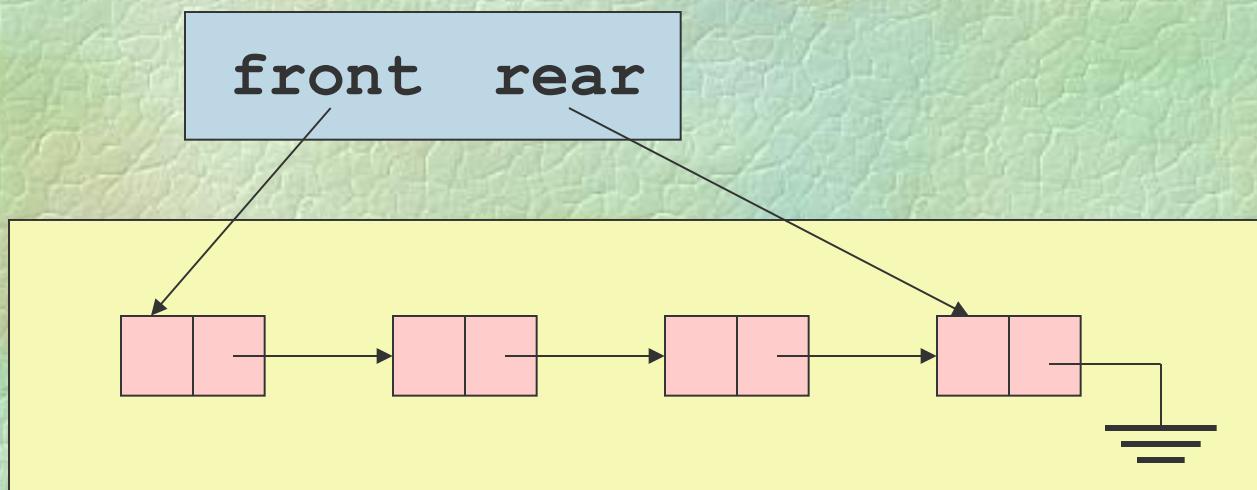
- First In First Out (FIFO)  
ό,τι μπαίνει πρώτο, βγαίνει πρώτο



### □ Αφηρημένος τύπος δεδομένων

- Ορίζεται ο τύπος **queue** που υλοποιεί την ουρά (ακεραίων αριθμών)
  - Ορίζονται οι απαραίτητες πράξεις:
    - **(queue)** δημιουργεί μια άδεια ουρά
    - **empty** ελέγχει αν μια ουρά είναι άδεια
    - **enqueue** προσθήκη στοιχείου στο τέλος
    - **dequeue** αφαίρεση στοιχείου από την αρχή

- Υλοποίηση με απλά συνδεδεμένη λίστα



- Υλοποίηση με απλά συνδεδεμένη λίστα

```
class queue {  
public:  
    queue ();  
    bool empty ();  
    void enqueue (int x);  
    int dequeue ();  
  
private:  
    struct node {  
        int info;  
        node *next;  
    };  
    node *front, *rear;  
};
```

## □ Κατασκευή άδειας ουράς

```
queue::queue () {  
    front = rear = nullptr;  
}
```

## □ Έλεγχος αν μια ουρά είναι άδεια

```
bool queue::empty () {  
    return front == nullptr;  
}
```

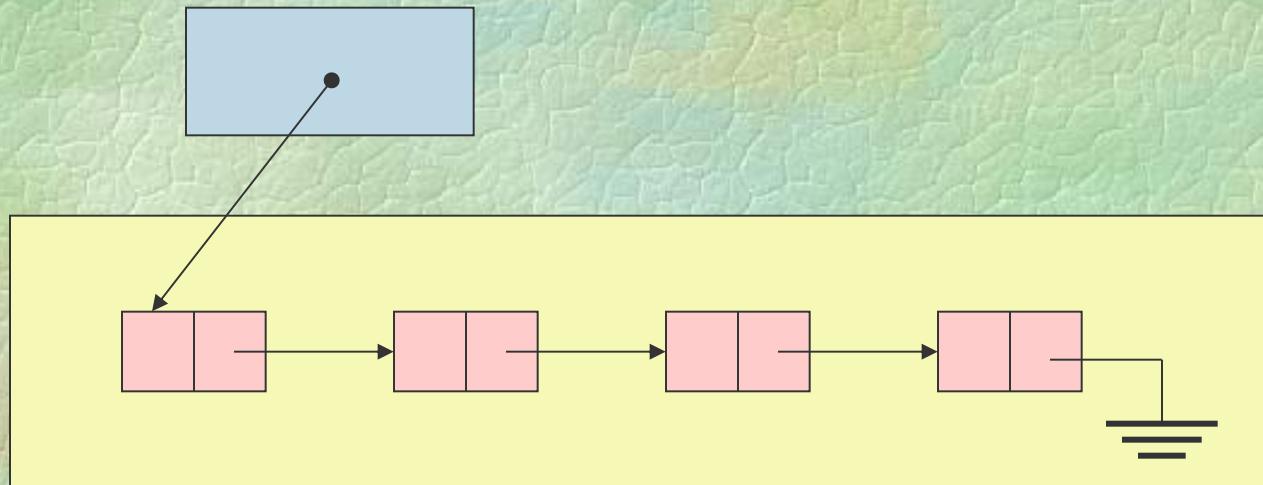
## □ Προσθήκη στοιχείου

```
void queue::enqueue (int x) {  
    node *p = new node;  
    p->info = x;  
    p->next = nullptr;  
    if (front == nullptr)  
        front = p;  
    else  
        rear->next = p;  
    rear = p;  
}
```

## □ Αφαίρεση στοιχείου

```
int queue::dequeue () {  
    node *p = front;  
    int result = front->info;  
    if (front == rear)  
        rear = nullptr;  
    front = front->next;  
    delete p;  
    return result;  
}
```

## □ Γενική μορφή απλά συνδεδεμένης λίστας



```
struct node {  
    int info;  
    node *next;  
};  
  
typedef node *list;
```

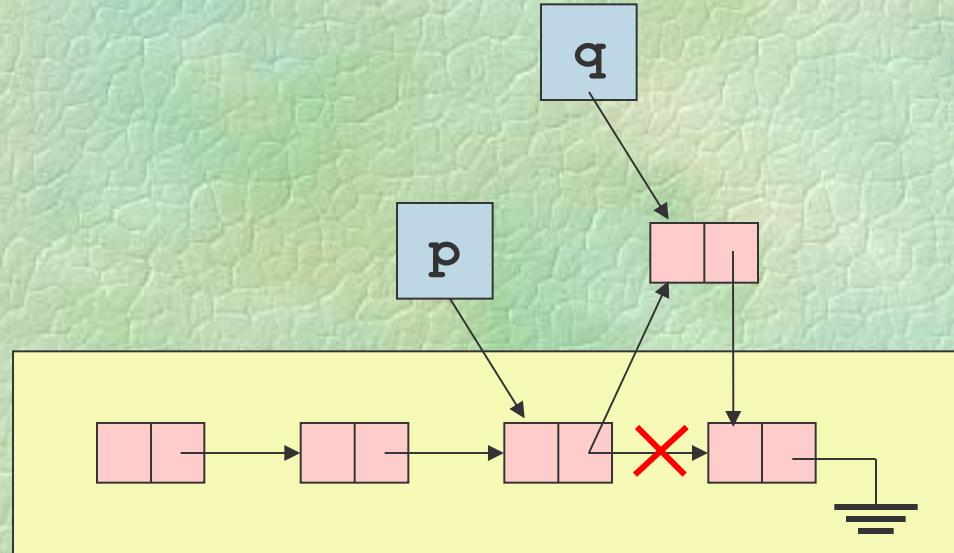
- Εισαγωγή στο τέλος O(n)

```
void insertAtRear (list &l, int data) {  
    node *p, *q;  
  
    p = new node;  
    p->info = data; p->next = nullptr;  
    if (l == nullptr) l = p;  
    else {  
        q = l;  
        while (q->next != nullptr)  
            q = q->next;  
        q->next = p;  
    }  
}
```

- Εισαγωγή μετά τον κόμβο p

O(1)

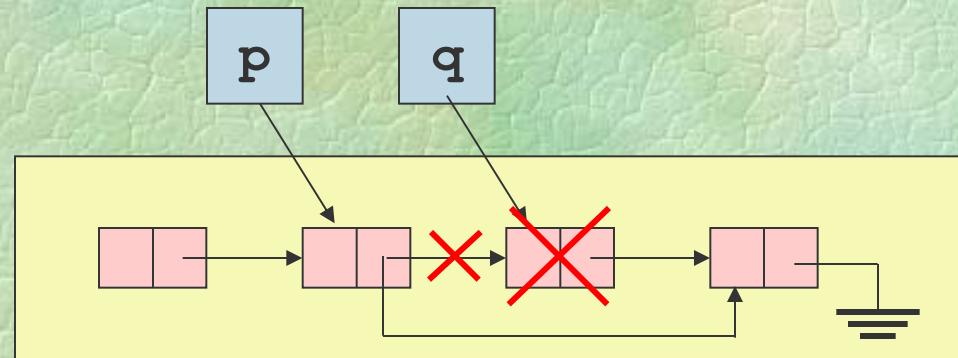
```
void insertAfter (node *p, int data) {  
    if (p != nullptr) {  
        node *q = new node;  
        q->info = data;  
        q->next = p->next;  
        p->next = q;  
    }  
}
```



- Διαγραφή του κόμβου μετά τον p

O(1)

```
void deleteAfter (node *p) {  
    if (p != nullptr &&  
        p->next != nullptr) {  
        node *q = p->next;  
        p->next = q->next;  
        delete q;  
    }  
}
```



## □ Εύρεση στοιχείου

$O(n)$

```
node *search (list l, int data) {  
    node *p;  
  
    for (p = l; p != nullptr; p = p->next)  
        if (p->info == data) return p;  
    return nullptr;  
}
```

- Συνένωση δύο λιστών  $O(n)$

```
void concat (list &l1, list l2) {  
    node *p;  
  
    if (l2 == nullptr) return;  
    if (l1 == nullptr) l1 = l2;  
    else {  
        p = l1;  
        while (p->next != nullptr)  
            p = p->next;  
        p->next = l2;  
    }  
}
```

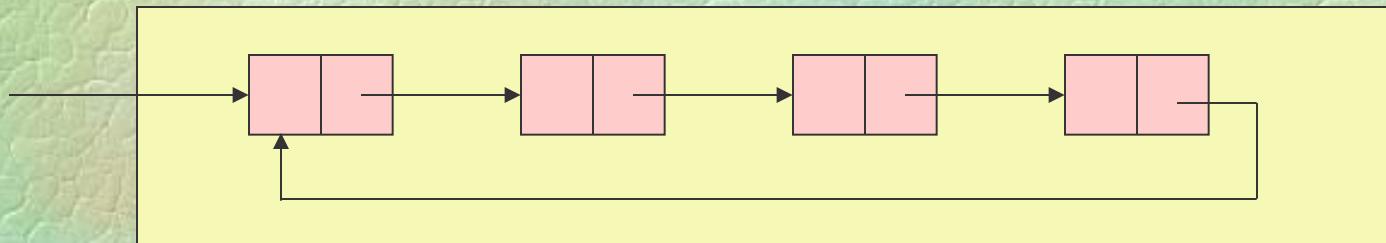
## □ Αντιστροφή λίστας

$O(n)$

```
void reverse (list &l) {  
    node *p, *q;  
  
    q = nullptr;  
    while (l != nullptr) {  
        p = l;  
        l = p->next;  
        p->next = q;  
        q = p;  
    }  
    l = q;  
}
```

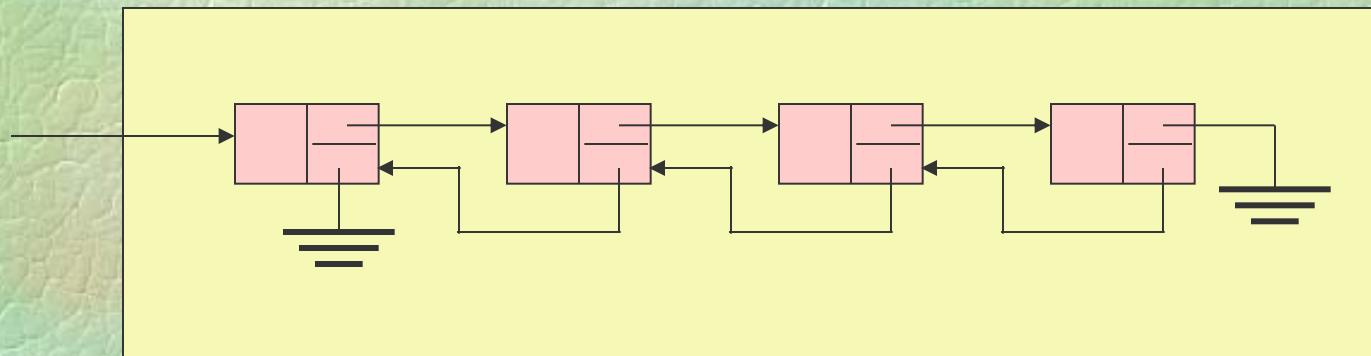
# Κυκλικές λίστες

- Ο επόμενος του τελευταίου κόμβου είναι πάλι ο πρώτος



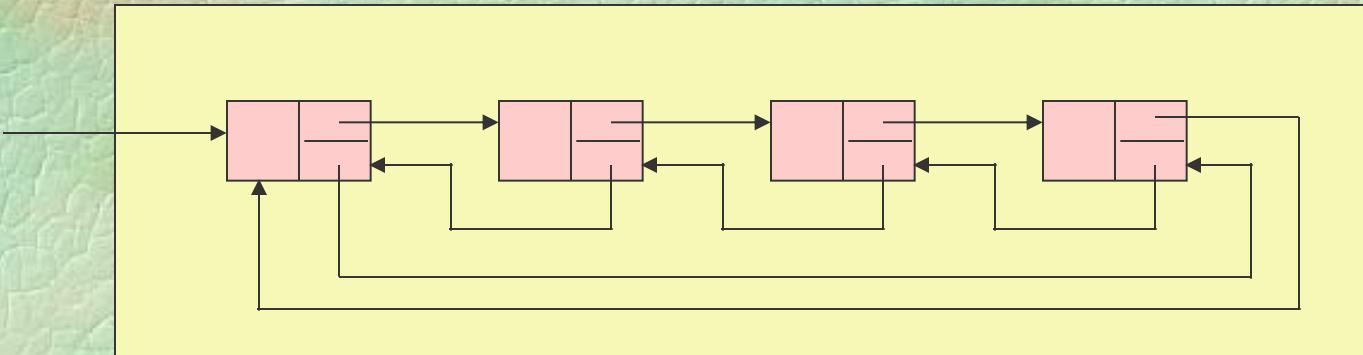
# Διπλά συνδεδεμένες λίστες

- Δυο σύνδεσμοι σε κάθε κόμβο, προς τον επόμενο και προς τον προηγούμενο



# Διπλά συνδεδεμένες κυκλικές λίστες

- Δύο σύνδεσμοι σε κάθε κόμβο, προς τον επόμενο και προς τον προηγούμενο
- Ο επόμενος του τελευταίου είναι ο πρώτος
- Ο προηγούμενος του πρώτου είναι ο τελευταίος



□ Γράφος ή γράφημα (graph)       $G = (V, E)$

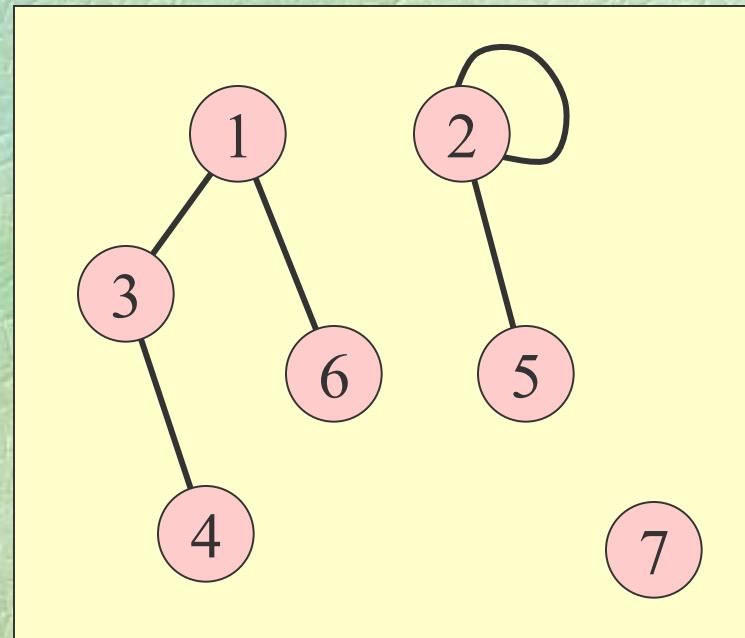
- $V$  Σύνολο κόμβων ή κορυφών
- $E$  Σύνολο ακμών, δηλαδή ζευγών κόμβων

□ Παράδειγμα

$$V = \{ 1, 2, 3, 4, 5, 6, 7 \}$$

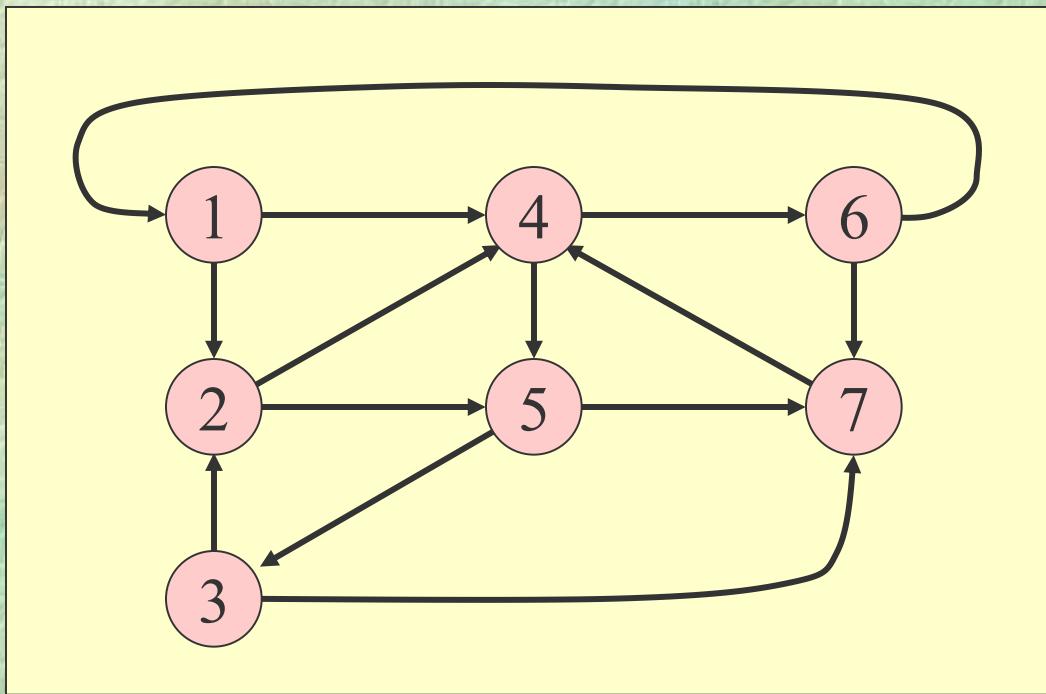
$$E = \{ (x, y) \mid x, y \in V, \\ x+y=4 \text{ ή } x+y=7 \}$$

□ Γραφική παράσταση



## □ Κατευθυνόμενος γράφος (directed graph)

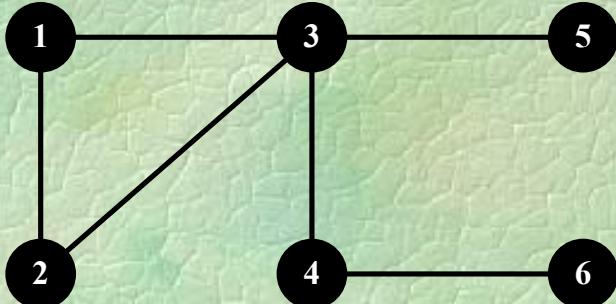
- Οι ακμές είναι διατεταγμένα ζεύγη
- Μπορούν να υλοποιηθούν με δείκτες



# Αναπαράσταση Γράφων

... με **πίνακα γειτνίασης**:  $A[i, j] = \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & (v_i, v_j) \notin E \end{cases}$

- Αν έχουμε βάρη,  $A[i, j] = w(v_i, v_j)$
- Μη-κατευθυνόμενο: **συμμετρικός** πίνακας.
- Χώρος  $\Theta(n^2)$ , άμεσος έλεγχος για ύπαρξη ακμής.

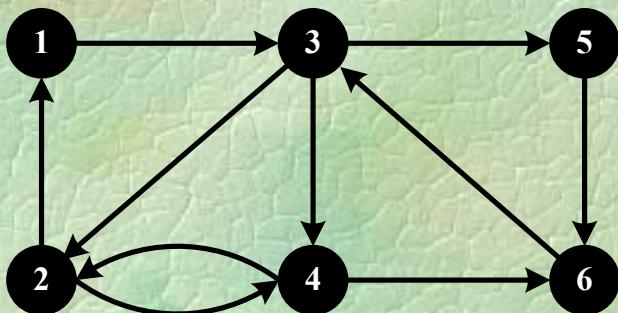


|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 0 | 0 | 0 |
| 3 | 1 | 1 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 0 | 1 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 1 | 0 | 0 |

# Αναπαράσταση Γράφων

... με **πίνακα γειτνίασης**:  $A[i, j] = \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & (v_i, v_j) \notin E \end{cases}$

- Αν έχουμε βάρη,  $A[i, j] = w(v_i, v_j)$
- Μη-κατευθυνόμενο: **συμμετρικός** πίνακας.
- Χώρος  $\Theta(n^2)$ , άμεσος έλεγχος για ύπαρξη ακμής.

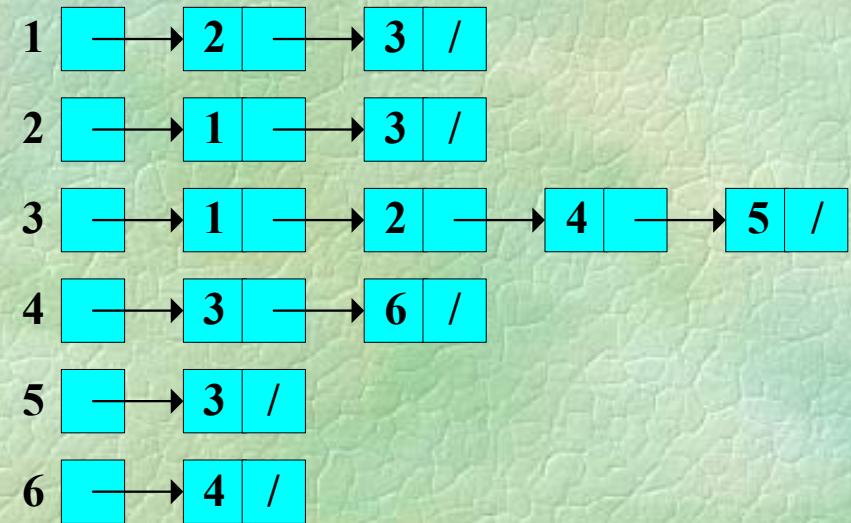
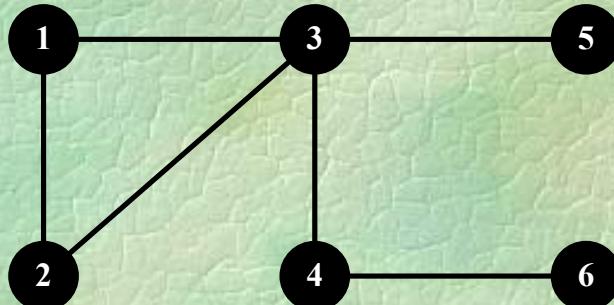


|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 |
| 6 | 0 | 0 | 1 | 0 | 0 | 0 |

# Αναπαράσταση Γράφων

... με λίστα γειτνίασης: γειτονικές κορυφές σε διασυνδεδεμένη λίστα.

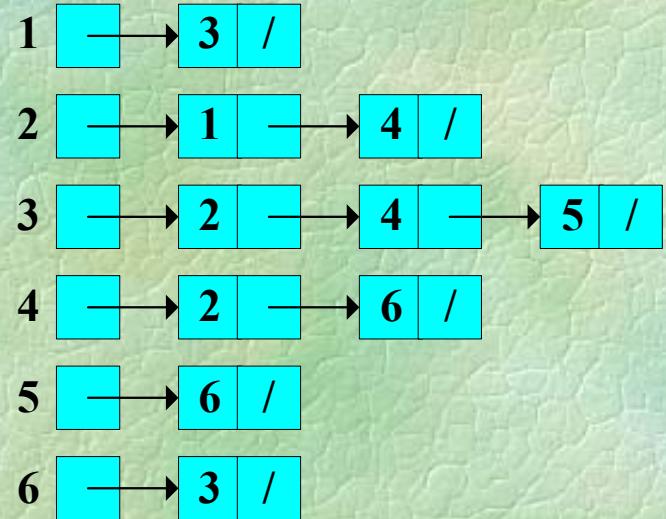
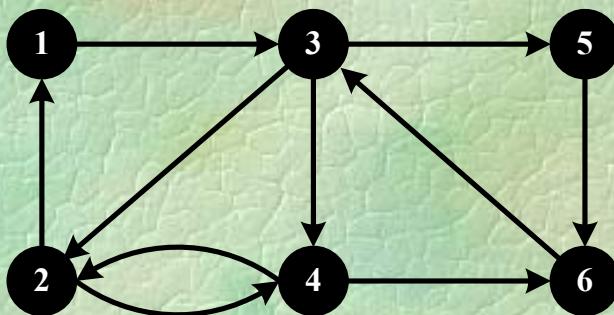
- Βάρη αποθηκεύονται στους κόμβους της λίστας.
- Χώρος  $\Theta(m)$ .
- Έλεγχος για ύπαρξη ακμής σε χρόνο  $O(\text{βαθμός}(u))$ .



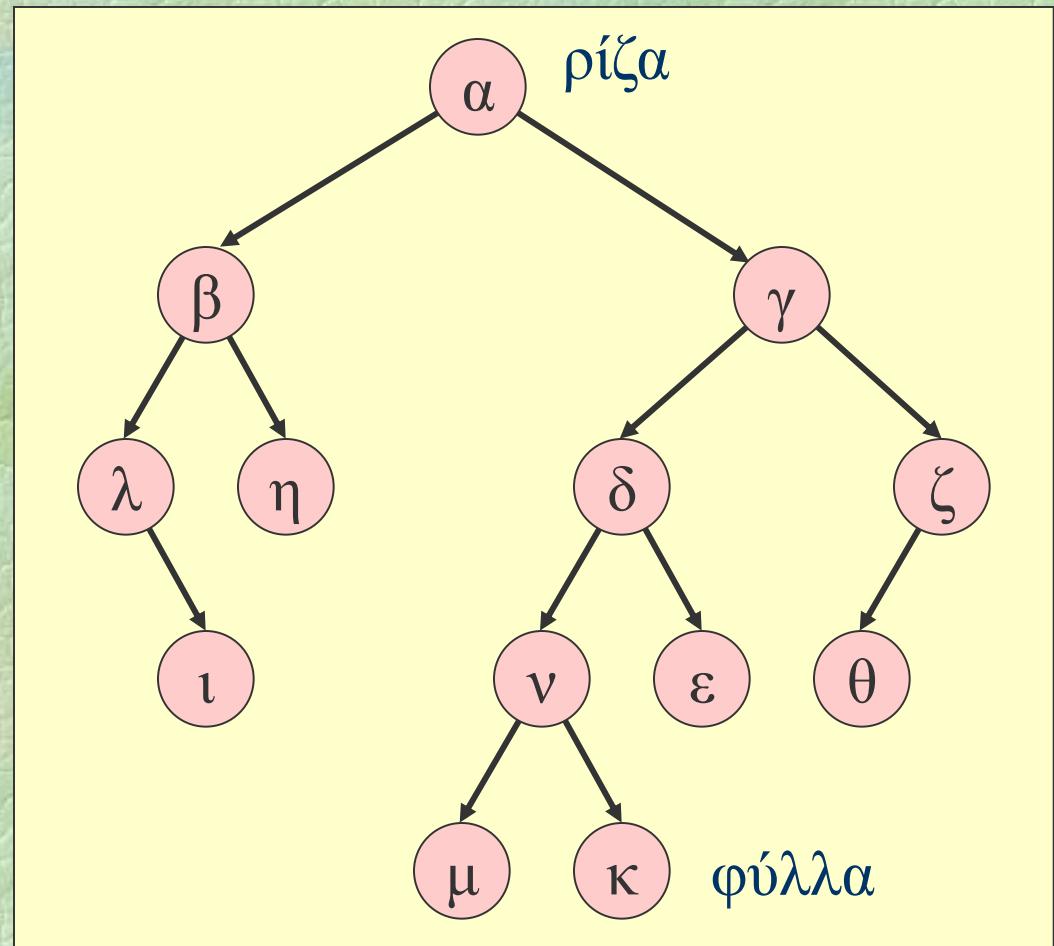
# Αναπαράσταση Γράφων

... με λίστα γειτνίασης: γειτονικές κορυφές σε διασυνδεδεμένη λίστα.

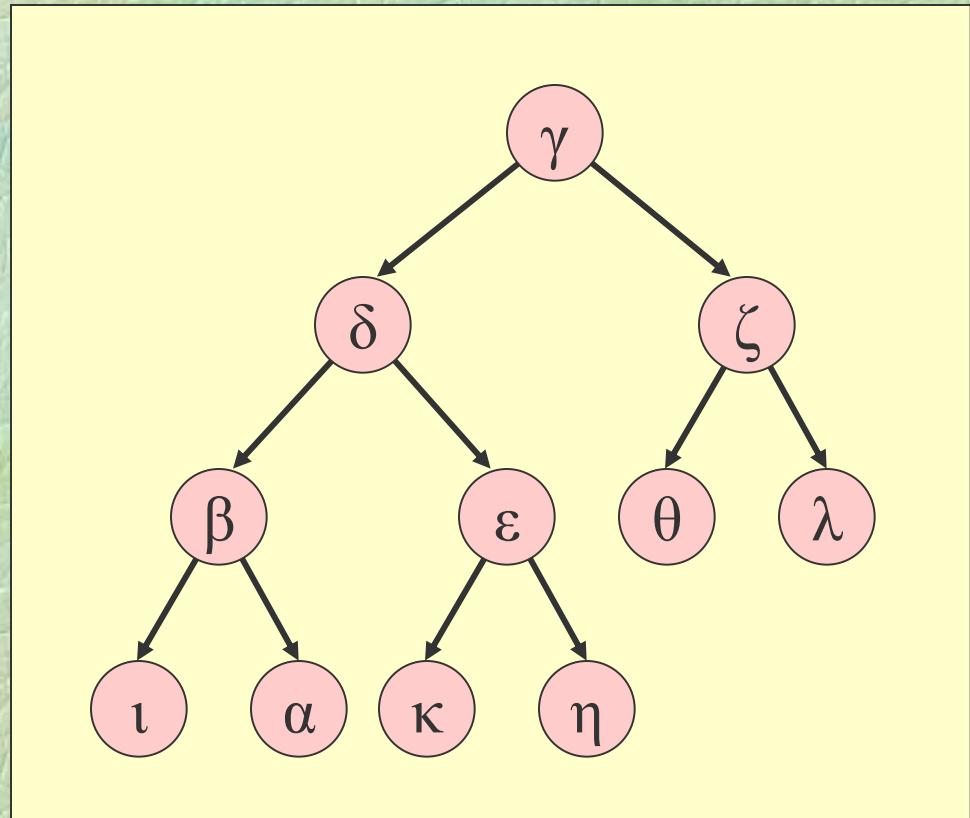
- Βάρη αποθηκεύονται στους κόμβους της λίστας.
- Χώρος  $\Theta(m)$ .
- Έλεγχος για ύπαρξη ακμής σε χρόνο  $O(\deg(u))$ .



- Ειδικοί γράφοι της μορφής:
- Κάθε κόμβος έχει 0, 1 ή 2 παιδιά
- Ρίζα: ο αρχικός κόμβος του δένδρου
- Φύλλα: κόμβοι χωρίς παιδιά

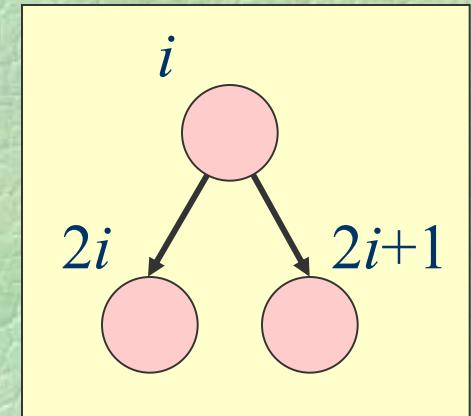


- Πλήρες δυαδικό δέντρο:
- Μόνο το κατώτατο επίπεδο μπορεί να μην είναι πλήρες
- Πλήθος κόμβων =  $n \Rightarrow$  ύψος =  $O(\log n)$



## □ Υλοποίηση με πίνακα

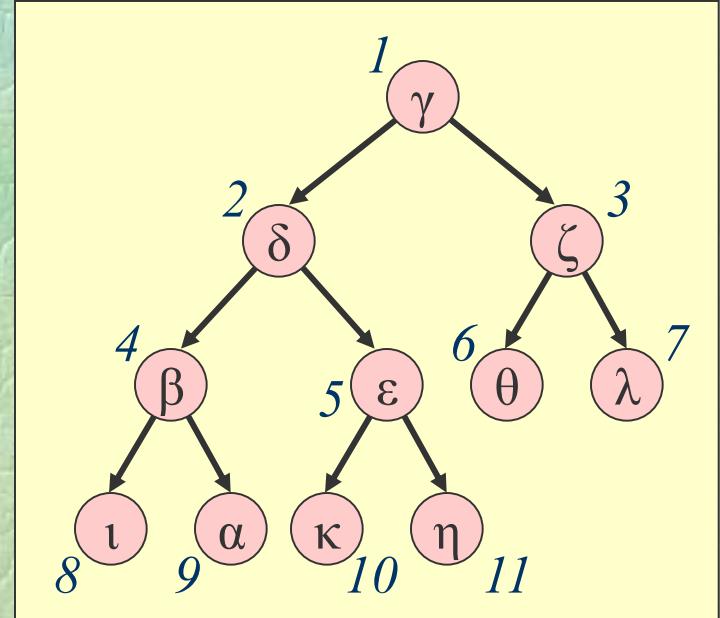
- Αν ένας κόμβος αποθηκεύεται στη θέση  $i$  του πίνακα, τα παιδιά του αποθηκεύονται στις θέσεις  $2i$  και  $2i+1$



## □ Παράδειγμα

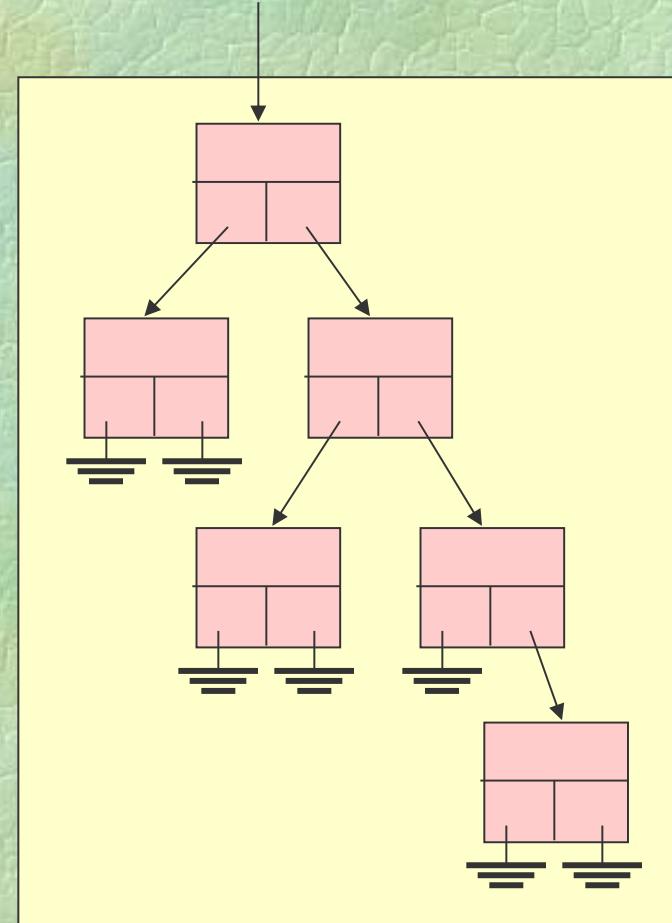
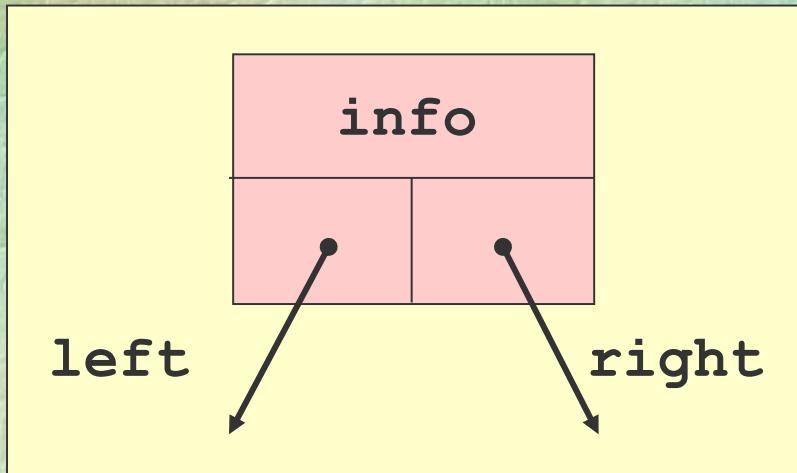
```

a[1] = 'γ'; a[7] = 'λ';
a[2] = 'δ'; a[8] = 'ι';
a[3] = 'ζ'; a[9] = 'α';
a[4] = 'β'; a[10] = 'κ';
a[5] = 'ε'; a[11] = 'η';
a[6] = 'θ'
  
```



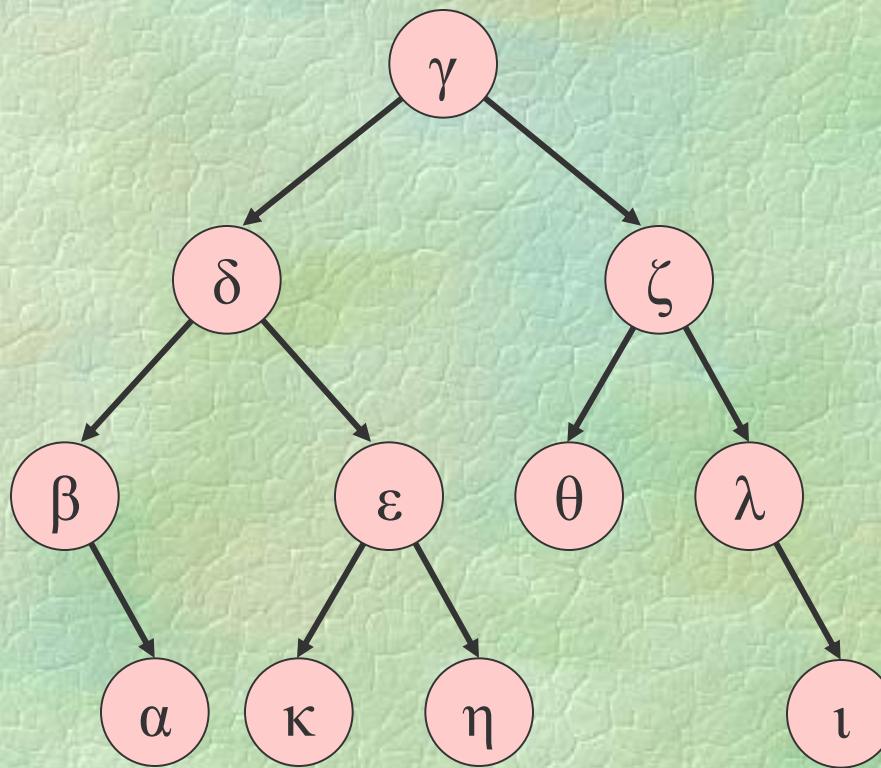
### □ Υλοποίηση με δείκτες

```
struct node {  
    int info;  
    node *left, *right;  
};  
► typedef node *tree;
```

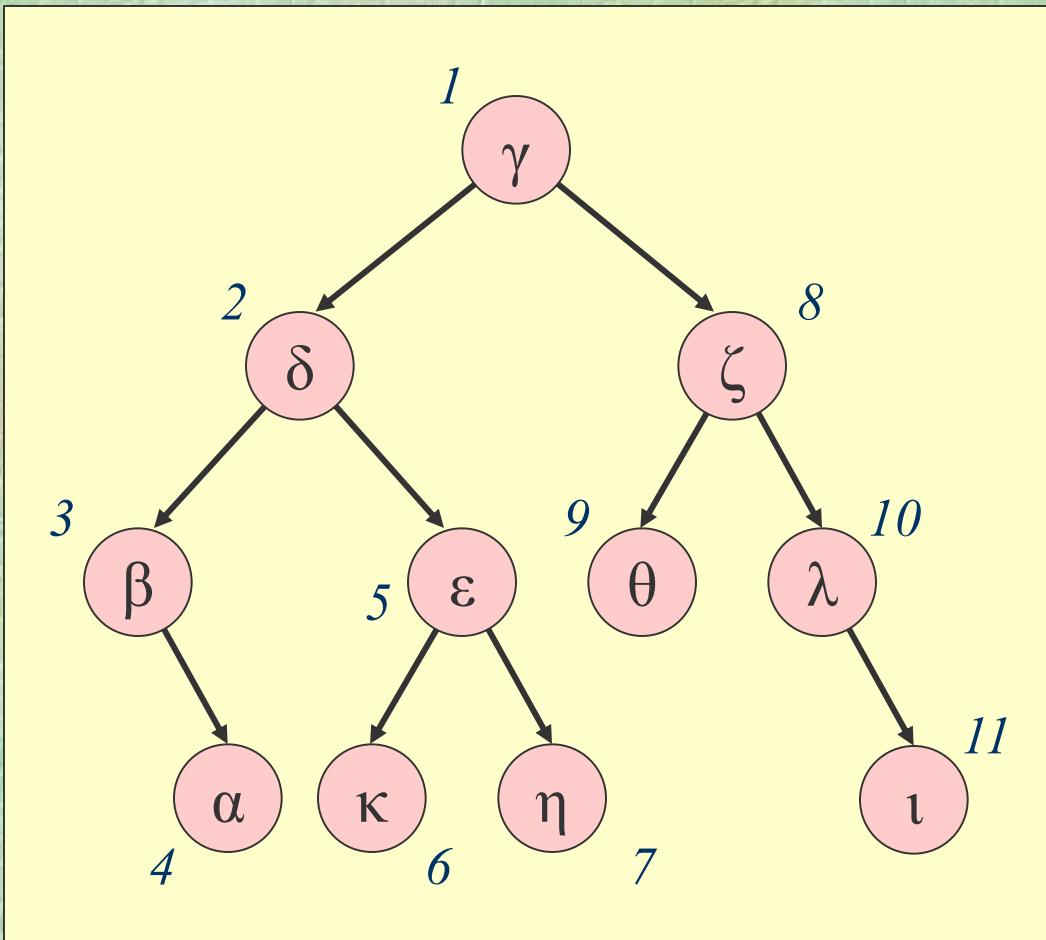


- Διάσχιση όλων των κόμβων ενός δέντρου
  - προθεματική διάταξη (**preorder**)  
για κάθε υποδέντρο, πρώτα η ρίζα,  
μετά το αριστερό υποδέντρο και μετά το δεξιό
  - επιθεματική διάταξη (**postorder**)  
για κάθε υποδέντρο, πρώτα το αριστερό  
υποδέντρο, μετά το δεξιό και μετά η ρίζα
  - ενθεματική διάταξη (**inorder**)  
για κάθε υποδέντρο, πρώτα το αριστερό  
υποδέντρο, μετά η ρίζα και μετά το δεξιό

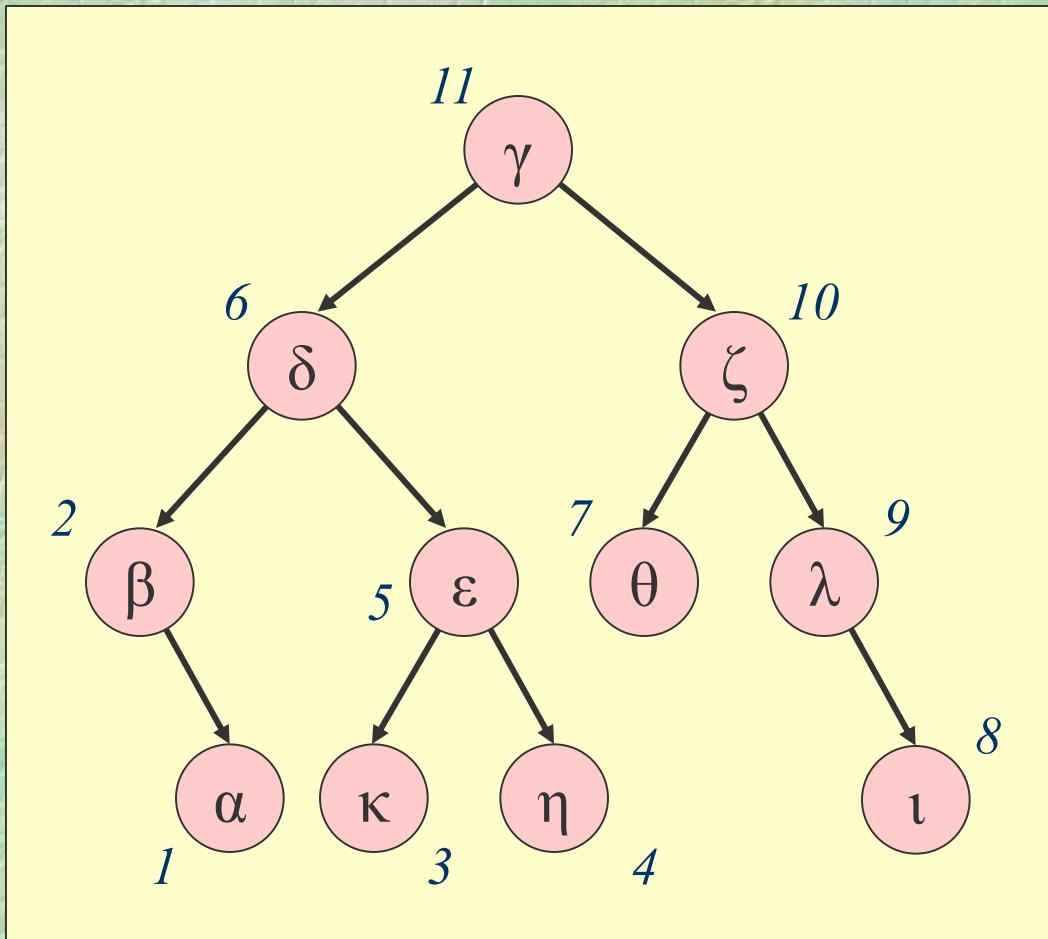
- Διάσχιση preorder / inorder / postorder



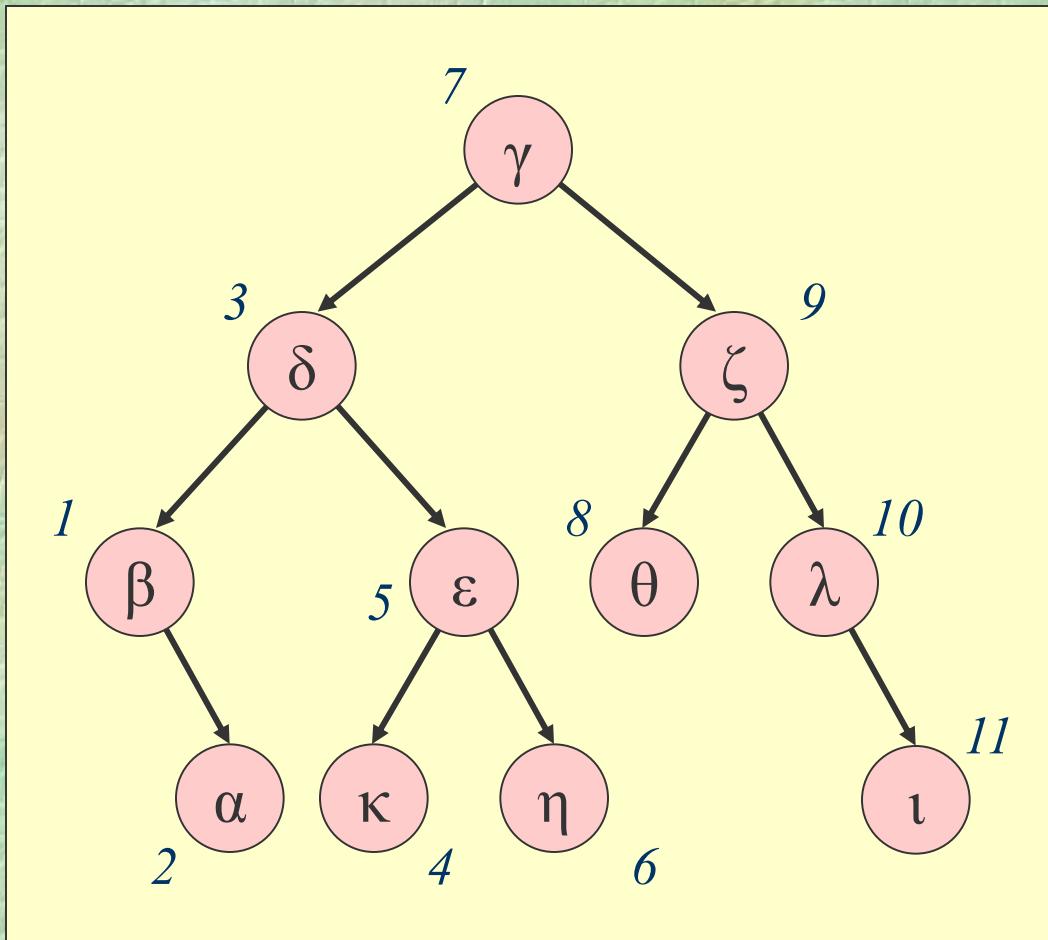
## □ Διάσχιση preorder



## □ Διάσχιση postorder



## □ Διάσχιση inorder



- Υλοποίηση της διάσχισης preorder

```
void preorder (tree t) {  
    if (t != nullptr) { WRITELN(t->info);  
        preorder(t->left);  
        preorder(t->  
                right);  
    }  
}
```

- Η παραπάνω διαδικασία είναι αναδρομική
- Η μη αναδρομική διάσχιση είναι εφικτή αλλά πολύπλοκη (threading)

## □ Πλήθος κόμβων και ύψος δέντρου

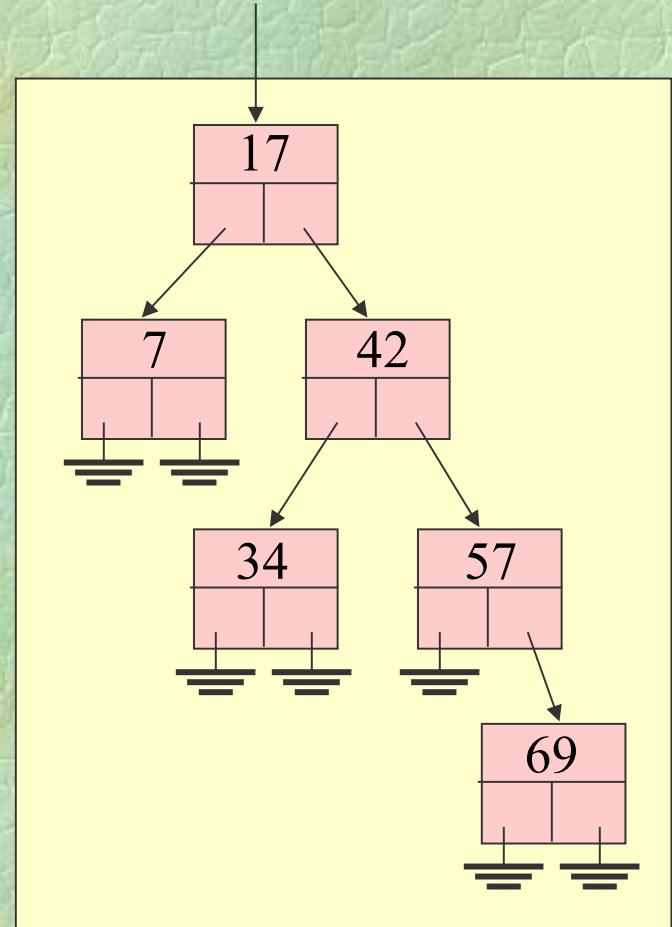
```
int size(tree t) {  
    if (t == nullptr) return 0;  
    return 1 + size(t->left)  
        + size(t->right);  
}  
  
int height(tree t) {  
    if (t == nullptr) return 0;  
    return 1 + max(height(t->left),  
                    height(t->right));  
}
```

Πολυπλοκότητα;  
 $O(n)$

# Δυαδικά δέντρα αναζήτησης

(i)

- Binary search trees
- Για κάθε κόμβο ισχύουν οι παρακάτω ιδιότητες:
  - όλοι οι κόμβοι του αριστερού υποδέντρου έχουν τιμές **μικρότερες** ή **ίσες** της τιμής του κόμβου
  - όλοι οι κόμβοι του δεξιού υποδέντρου έχουν τιμές **μεγαλύτερες** ή **ίσες** της τιμής του κόμβου



- Τα δυαδικά δέντρα αναζήτησης διευκολύνουν την αναζήτηση στοιχείων
- Αναδρομική αναζήτηση
  - αν η τιμή που ζητείται είναι στη ρίζα, βρέθηκε
  - αν είναι μικρότερη από την τιμή της ρίζας, αρκεί να αναζητηθεί στο αριστερό παιδί
  - αν είναι μεγαλύτερη από την τιμή της ρίζας, αρκεί να αναζητηθεί στο δεξί παιδί
- Κόστος αναζήτησης, εισαγωγής, διαγραφής:  $O(k)$ , όπου  $k$  είναι το ύψος του δέντρου

## □ Αναζήτηση

```
node *search (node *t, int key) {  
    if (t == nullptr)  
        return nullptr; // not found  
  
    if (t->info == key) return t; // found  
  
    if (t->info > key)  
        return search(t->left, key);  
    else  
        return search(t->right, key);  
}
```

### □ Εισαγωγή

```
node *insert(node *t, int key) {  
    if (t == nullptr) {  
        node *p = new node;  
        p->info = key;  
        p->left = p->right = nullptr;  
        return p;  
    }  
    if (t->info > key)  
        t->left = insert(t->left, key);  
    else if (t->info < key)  
        t->right = insert(t->right, key);  
    return t;  
}
```

## □ Ισοζύγισμα

- Εφόσον το κόστος των πράξεων είναι  $O(k)$  θέλουμε να κρατάμε μικρό το ύψος  $k$  του δέντρου
- **Ισοζυγισμένα δένδρα** (balanced trees): το βάθος δυο φύλλων διαφέρει το πολύ κατά 1
- Διάφορες παραλλαγές ορίζουν διαφορετικά την έννοια του ισοζυγίσματος: AVL, red-black trees
- Σε ένα ισοζυγισμένο δυαδικό δέντρο αναζήτησης με  $n$  κόμβους, μπορούμε να βρεθούμε από τη ρίζα σε οποιοδήποτε κόμβο με κόστος  $O(\log n)$

# Το λειτουργικό σύστημα Unix (i)

- Bell Labs, ~1970
- Δομή του Unix
  - πυρήνας (kernel)
  - φλοιός (shell)
  - βοηθητικά προγράμματα (utilities)
- Ιεραρχικό σύστημα αρχείων
  - Δενδρική δομή
  - Ένας κατάλογος (directory) μπορεί να περιέχει αρχεία (files) ή άλλους (υπο)καταλόγους

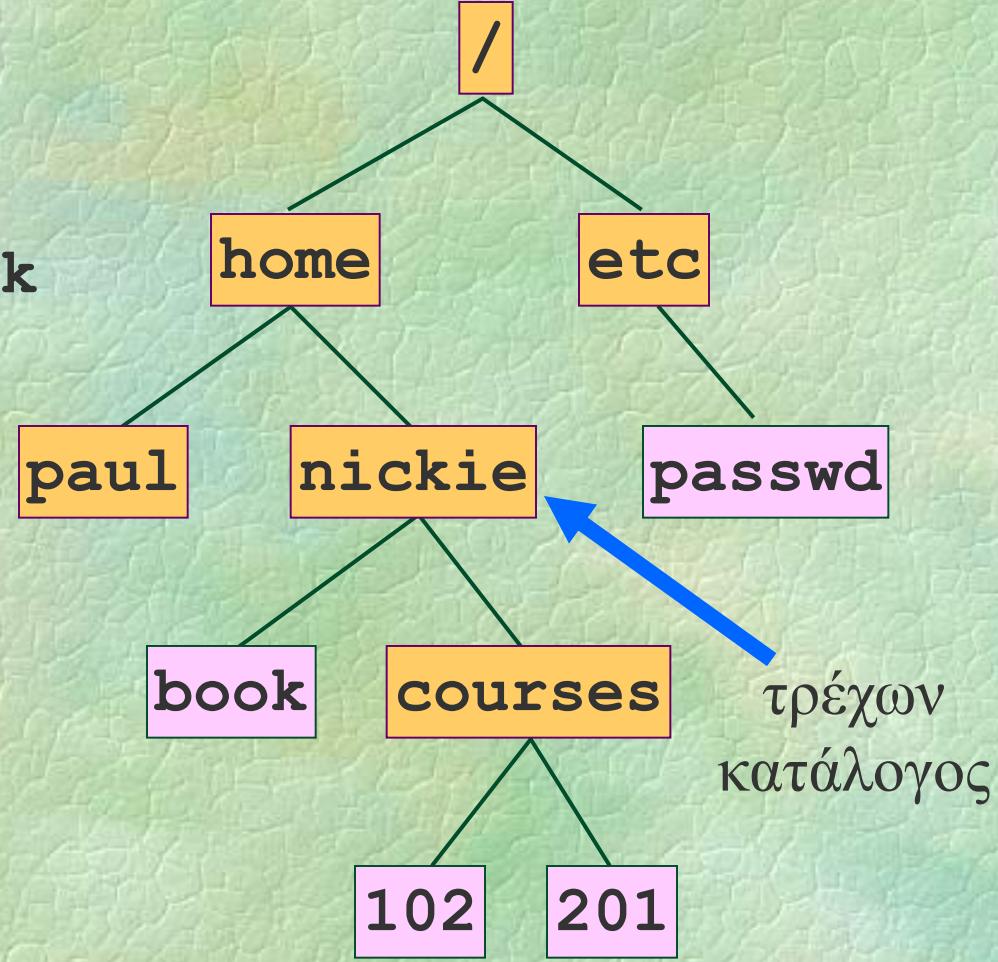
# Το λειτουργικό σύστημα Unix (ii)

## □ Απόλυτα ονόματα

/  
/etc  
/home/nickie/book  
/home/paul  
/etc/passwd

## □ Σχετικά ονόματα

book  
courses/201  
. /courses/102  
. . /paul  
. . . /etc/passwd



# Το λειτουργικό σύστημα Unix (iii)

## □ Θετικά στοιχεία του Unix

- Ιεραρχικό σύστημα αρχείων
- πολλοί χρήστες συγχρόνως (multi-user)
- πολλές διεργασίες συγχρόνως (multi-tasking)
- επικοινωνίες και υποστήριξη δικτύου

## □ Αρνητικά στοιχεία του Unix

- κρυπτογραφικά ονόματα εντολών
- περιορισμένη και συνθηματική βοήθεια

# Σύστημα αρχείων του Unix (i)

## □ Αντιγραφή αρχείων

**cp**

**cp** *oldfile newfile*

**cp** *file<sub>1</sub> file<sub>2</sub> ... file<sub>n</sub> directory*

**cp -r** *directory<sub>1</sub> directory<sub>2</sub>*

**cp -i** *oldfile newfile*

## □ Μετονομασία ή μετακίνηση αρχείων

**mv**

**mv** *oldfile newfile*

**mv** *file<sub>1</sub> file<sub>2</sub> ... file<sub>n</sub> directory*

**mv -i** *oldfile newfile*

# Σύστημα αρχείων του Unix (ii)

## □ Διαγραφή αρχείων

**rm**

**rm** *file<sub>1</sub>* *file<sub>2</sub>* ... *file<sub>n</sub>*  
**rm** **-i** *file<sub>1</sub>* *file<sub>2</sub>* ... *file<sub>n</sub>*  
**rm** **-f** *file<sub>1</sub>* *file<sub>2</sub>* ... *file<sub>n</sub>*  
**rm** **-r** *directory*

## □ Δημιουργία directories

**mkdir**

**mkdir** *directory<sub>1</sub>* ... *directory<sub>n</sub>*

## □ Διαγραφή άδειων directories

**rmdir**

**rmdir** *directory<sub>1</sub>* ... *directory<sub>n</sub>*

## □ Αλλαγή directory

**cd**

**cd** *directory*

## □ Εμφάνιση πληροφοριών για αρχεία

**ls**

**ls**

**ls** *file<sub>1</sub>* *file<sub>2</sub>* *directory<sub>3</sub>* . . .

### □ Επιλογές (options)

**-l** εκτεταμένες πληροφορίες

**-a** εμφανίζονται και τα κρυφά αρχεία

**-t** ταξινόμηση ως προς το χρόνο τροποποίησης

**-F** εμφανίζεται ο τύπος κάθε αρχείου

**-d** εμφανίζονται πληροφορίες για ένα directory,  
όχι για τα περιεχόμενά του

**-R** αναδρομική εμφάνιση πληροφοριών

# Προγράμματα εφαρμογών Unix (i)

- Εμφάνιση manual page **man**

**man** *command*

**whatis** *command*

- Εμφάνιση περιεχομένων αρχείου **cat**

**cat** *file<sub>1</sub>* *file<sub>2</sub>* ... *file<sub>n</sub>*

- Εμφάνιση περιεχομένων αρχείου ανά σελίδα **more**  
**less**

**more** *file<sub>1</sub>* *file<sub>2</sub>* ... *file<sub>n</sub>*

**less** *file<sub>1</sub>* *file<sub>2</sub>* ... *file<sub>n</sub>*

# Προγράμματα εφαρμογών Unix (ii)

- Εμφάνιση πρώτων γραμμών **head**  
**head**  $file_1 \ file_2 \ \dots \ file_n$   
**head -10**  $file_1 \ file_2 \ \dots \ file_n$
- Εμφάνιση τελευταίων γραμμών **tail**  
**tail**  $file_1 \ file_2 \ \dots \ file_n$   
**tail -10**  $file_1 \ file_2 \ \dots \ file_n$
- Πληροφορίες για το είδος αρχείου **file**  
**file**  $file_1 \ file_2 \ \dots \ file_n$
- Εμφάνιση ημερομηνίας και ώρας **date**  
**date**

# Προγράμματα εφαρμογών Unix (iii)

## □ Εκτύπωση αρχείου

**lpr** *file<sub>1</sub> file<sub>2</sub> ... file<sub>n</sub>*

**lpr**

## □ Μεταγλωττιστής Pascal

**pc** -o *executable program.p*

**pc**

**gpc** -o *executable program.p*

## □ Μεταγλωττιστής C

**cc** -o *executable program.p*

**cc**

**gcc** -o *executable program.p*

## □ Επεξεργασία αρχείου κειμένου

**vi** *file<sub>1</sub> file<sub>2</sub> ... file<sub>n</sub>*

**vi**

# Βασική λειτουργία του vi (i)

- Δύο καταστάσεις λειτουργίας
  - κατάσταση εντολών
  - κατάσταση εισαγωγής κειμένου
- Στην κατάσταση εισαγωγής κειμένου
  - πηγαίνουμε με συγκεκριμένες εντολές (π.χ. **i**, **a**)
  - μπορούμε μόνο να εισάγουμε χαρακτήρες
- Στην κατάσταση εντολών
  - πηγαίνουμε με το πλήκτρο **ESC**
  - μπορούμε να μετακινούμαστε και να δίνουμε εντολές

# Βασική λειτουργία του vi (ii)

## □ Μετακίνηση μέσα σε αρχείο

← ↓ ↑ → κατά ένα χαρακτήρα

h j k l (ομοίως)

w μια λέξη δεξιά

CTRL+F μια σελίδα μετά

CTRL+B μια σελίδα πριν

CTRL+D μισή σελίδα μετά

CTRL+U μισή σελίδα πριν

0 \$ στην αρχή ή στο τέλος της γραμμής

^ στον πρώτο χαρακτήρα της γραμμής

# Βασική λειτουργία του vi (iii)

## □ Μετακίνηση μέσα σε αρχείο (συνέχεια)

- + στην αρχή της προηγούμενης ή  
της επόμενης γραμμής

( ) στην αρχή της προηγούμενης ή  
της επόμενης πρότασης

{ } στην αρχή της προηγούμενης ή  
της επόμενης παραγράφου

*n* G στην *n*-οστή γραμμή

G στην τελευταία γραμμή

## □ Εισαγωγή κειμένου

**i** α εισαγωγή πριν ή μετά τον cursor

**I A** εισαγωγή στην αρχή ή στο τέλος της γραμμής

**o O** εισαγωγή σε νέα κενή γραμμή κάτω ή πάνω από την τρέχουσα

**r** αντικατάσταση ενός χαρακτήρα

**R** αντικατάσταση πολλών χαρακτήρων

# Βασική λειτουργία του vi (v)

## □ Διαγραφή κειμένου

- x      του τρέχοντα χαρακτήρα
- x      του προηγούμενου χαρακτήρα
- dw      μέχρι το τέλος λέξης
- dd      ολόκληρης της τρέχουσας γραμμής
- n dd n γραμμών αρχίζοντας από την  
τρέχουσα
- Οι λέξεις και οι γραμμές που διαγράφονται  
τοποθετούνται στο buffer (**cut**)

# Βασική λειτουργία του vi (vi)

## □ Εύρεση συμβολοσειράς

- / xxx εύρεση προς τα εμπρός
- ? xxx εύρεση προς τα πίσω
- n N επόμενη εύρεση ορθής ή  
αντίθετης φοράς

## □ Άλλες εντολές

- CTRL-L επανασχεδίαση της εικόνας
- u ακύρωση της τελευταίας εντολής
- . επανάληψη της τελευταίας εντολής
- J συνένωση της τρέχουσας γραμμής  
με την επόμενη

# Βασική λειτουργία του vi (vii)

## □ Αντιγραφή και μετακίνηση κειμένου

**yy** αντιγραφή μιας γραμμής στο buffer  
**(copy)**

**n yy** αντιγραφή **n** γραμμών στο buffer

**p P** επικόλληση των περιεχομένων του  
buffer κάτω ή πάνω από την τρέχουσα  
γραμμή **(paste)**

## □ Αποθήκευση και έξοδος

**:w** αποθήκευση του αρχείου

**:q** έξοδος

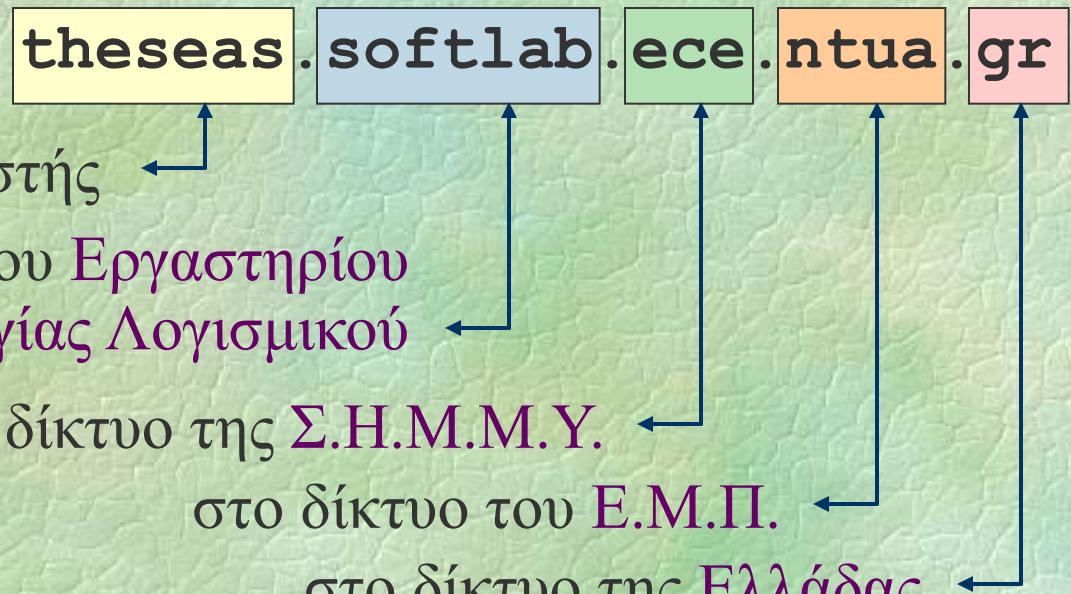
**:wq** αποθήκευση του αρχείου και έξοδος

**:q!** έξοδος χωρίς αποθήκευση

- Δίκτυο υπολογιστών (computer network)
- Ονόματα και διευθύνσεις υπολογιστών

□ Διεύθυνση IP    **147.102.1.1**

□ Όνομα



□ Επικράτειες  
(domains)

### □ Ηλεκτρονικό ταχυδρομείο (e-mail)

- ηλεκτρονική ταχυδρομική διεύθυνση



- υπάρχει πληθώρα εφαρμογών που διαχειρίζονται το ηλεκτρονικό ταχυδρομείο

- Πρόσβαση σε απομακρυσμένους υπολογιστές (telnet)

```
maya$ telnet theseas.softlab.ntua.gr
```

```
SunOS 5.7
```

```
login: nickie
```

```
Password:
```

```
Last login: Thu Jan 16 12:33:45
```

```
Sun Microsystems Inc. SunOS 5.7
```

```
You have new mail.
```

```
Fri Jan 17 03:16:45 EET 2003
```

```
There are 28 messages in your mailbox.
```

```
There are 2 new messages.
```

```
theseas$
```

- Μεταφορά αρχείων (FTP)
  - κατέβασμα αρχείων (download)  
μεταφορά αρχείων από τον απομακρυσμένο υπολογιστή προς τον τοπικό υπολογιστή
  - ανέβασμα αρχείων (upload)  
μεταφορά αρχείων από τον τοπικό υπολογιστή προς τον απομακρυσμένο υπολογιστή
  - anonymous FTP  
π.χ. **ftp.ntua.gr**

## □ Ηλεκτρονικά νέα (news)

- ομάδες συζήτησης (newsgroups)
  - η συζήτηση συνήθως περιστρέφεται γύρω από συγκεκριμένα θέματα
    - π.χ. **comp.lang.pascal**
  - οι ομάδες συζήτησης λειτουργούν σαν πίνακες ανακοινώσεων
  - καθένας μπορεί να διαβάζει τις ανακοινώσεις των άλλων και να βάλει την ανακοίνωσή του (posting)

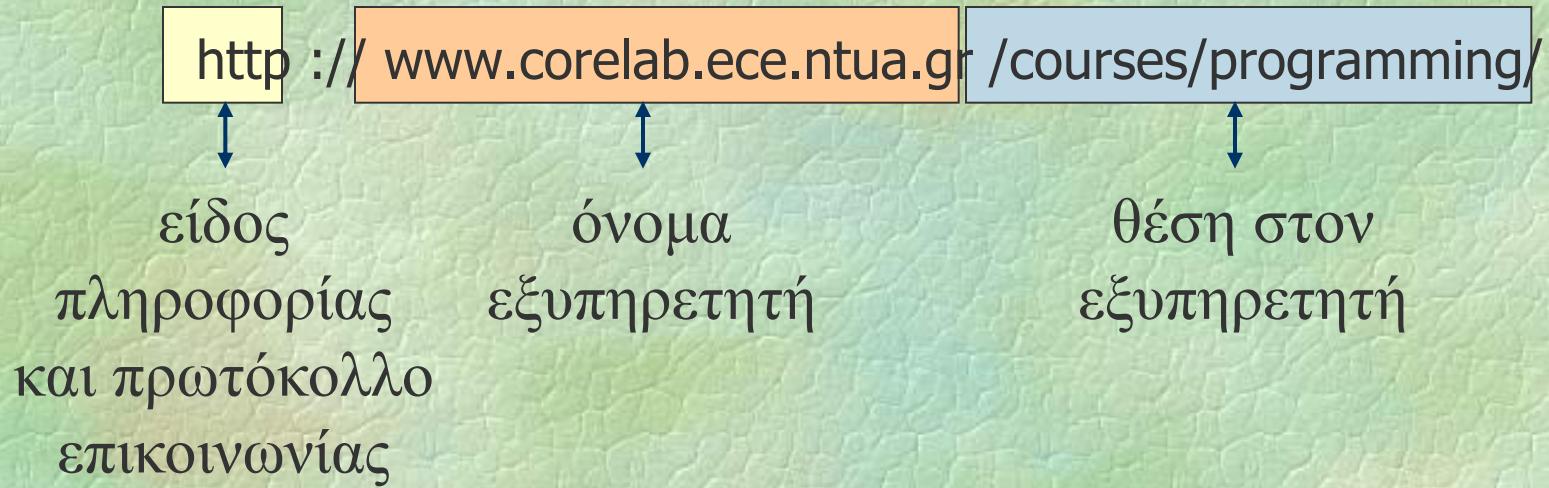
### □ Κουτσομπολιό (chat ή IRC)

- κανάλια (channels)
  - η συζήτηση περιστρέφεται γύρω από ένα θέμα κοινού ενδιαφέροντος
- είναι όμως σύγχρονη, δηλαδή γίνεται σε συγκεκριμένο χρόνο και δεν τηρείται αρχείο των λεχθέντων
- καθένας μπορεί να «ακούει» τα λεγόμενα των άλλων και να «μιλά» προς αυτούς

## □ Παγκόσμιος ιστός World-Wide Web (WWW)

- ένα σύστημα αναζήτησης υπερμεσικών πληροφοριών (hypermedia information)
- ιστοσελίδες (web pages), υπερμέσα (hypermedia), σύνδεσμοι (links), εξυπηρετητές (servers), και περιηγητές (browsers)

## □ Διευθύνσεις στον παγκόσμιο ιστό (URL)



# □ Παραδείγματα διευθύνσεων

<http://www.ntua.gr/>

<ftp://ftp.ntua.gr/pub/linux/README.txt>

[news://news.ntua.gr/comp.lang.pascal](http://news.ntua.gr/comp.lang.pascal)