

# ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

<http://courses.softlab.ntua.gr/progintro/>

Διδάσκοντες: Στάθης Ζάχος (zachos@cs.ntua.gr)  
Νίκος Παπασπύρου (nickie@softlab.ntua.gr)  
Άρης Παγουρτζής (pagour@cs.ntua.gr)

DRAFT

Διαφάνειες παρουσιάσεων

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

19/11/04

◆ Μάθημα: Προγραμματισμός Η/Υ

◆ Διδάσκοντες:

Σ. Ζάχος, Ν. Παπασπύρου, Α. Παγουρτζής

◆ Διαλέξεις: κάθε Παρασκευή 13:00–16:00

- Τμήμα Α: Αμφιθέατρο 1 (γράμματα Α–Λ)
- Τμήμα Β: Αμφιθέατρο 2 (γράμματα Μ–Ω)

◆ Εργαστήριο: βλ. πρόγραμμα εργαστηρίου

◆ Ιστοσελίδα του μαθήματος:

<http://courses.softlab.ntua.gr/progintro/>

## ◆ Τρόπος βαθμολογίας

- για όσους παίρνουν το μάθημα για πρώτη φορά
  - 7 μονάδες από την τελική εξέταση
  - 2 μονάδες από την πρόοδο
  - 2 μονάδες από το εργαστήριο
  - βάση: 5 στα 11 (συνολικά)  
και 2.3 στα 7 (στην τελική εξέταση)
- για σπουδαστές παλαιότερων εξαμήνων που  
έχουν ξαναπάρει το μάθημα
  - 10 μονάδες από την τελική εξέταση
  - βάση: 4.5 στα 10

## ◆ Εργαστήριο

- ④ *Ο κύριος τρόπος για να μάθει κανείς προγραμματισμό σε υπολογιστή*
- ④ *Η συμμετοχή στο εργαστήριο είναι απαραίτητη*
- Ανάλογα με το επώνυμό τους, οι σπουδαστές χωρίζονται σε 19 ομάδες εργαστηρίου
- Κάθε ομάδα έχει 2 ώρες την εβδομάδα
- Για κάθε ομάδα υπάρχει υπεύθυνος και βοηθοί
- Ελεύθερες ώρες για κάλυψη κενών, κ.λπ.
- Αρχίζει από τη Δευτέρα 11 Οκτωβρίου

## ◆ Πρόγραμμα εργαστηρίου

1A	ΔΕ 12:45–14:45	HMMY (α)	Αγγελης – Αντωνιου
1B	ΔΕ 12:45–14:45	HMMY (δ)	Ανυφαντης – Βορεαδη
1C	ΔΕ 12:45–14:45	Central (π)	Βουλγαρη – Γιαννέλος
1D	ΔΕ 12:45–14:45	Central (π)	Γιαννουλης – Ζαμκοτσιαν
2A	ΔΕ 15:00–17:00	HMMY (α)	Ζαμπαρα – Καλογεροπουλος
2B	ΔΕ 15:00–17:00	HMMY (δ)	Καλταμπανη – Καριανακης
2C	ΔΕ 15:00–17:00	Central (π)	Καρκατσουλη – Κολιοπουλου
2D	ΔΕ 15:00–17:00	Central (π)	Κολλια – Κουτσιανας
3A	ΤΡ 15:00–17:00	HMMY (α)	Κρεμιζας – Λιανου
3B	ΤΡ 15:00–17:00	HMMY (δ)	Λιαργκοβας – Μαρακη

◆ Πρόγραμμα εργαστηρίου (συνέχεια)

4A	ΤΕ 16:00–18:00	HMMY (α)	Μαραντης – Μπαρλα
4B	ΤΕ 16:00–18:00	HMMY (δ)	Μπαρμπουνακης – Ξυγκης
5A	ΠΕ 12:45–14:45	HMMY (α)	Οικονομου – Παπαικονομου
5B	ΠΕ 12:45–14:45	HMMY (δ)	Παπασπυρου – Πρεκας
5C	ΠΕ 12:45–14:45	Central (κ)	Ρεκατσινας – Σκιανη
6A	ΠΑ 09:30–11:30	HMMY (α)	Σκλαβουνου – Στεφανουδακη
6B	ΠΑ 09:30–11:30	HMMY (δ)	Στουμπη – Τσακωνας
6C	ΠΑ 09:30–11:30	Central (π)	Τσαμητρος – Φωτιας
6D	ΠΑ 09:30–11:30	Central (π)	Φωτιου – Ψωμαδακη

## ◆ Πρόσβαση στο εργαστήριο

- Υπολογιστής: courses.softlab.ntua.gr
- Χρειάζεστε login και password
- Αυτά είναι αυστηρά προσωπικά !

## ◆ Βαθμολογία στο εργαστήριο

- 1 μονάδα από ασκήσεις που παραδίδονται
- 1 μονάδα από εργαστηριακή εξέταση
- η βαθμολόγηση γίνεται από τους διδάσκοντες και τους υπευθύνους ομάδων

## ◆ Ασκήσεις που παραδίδονται

- **προγραμματιστικές ασκήσεις** (1, 2, 3, ...)   
 οι σπουδαστές σχεδιάζουν, κατασκευάζουν και εισάγουν στον Η/Υ ένα πρόγραμμα σύμφωνα με την εκφώνηση της άσκησης και το **επιδεικνύουν** (κείμενο σε Pascal και εκτέλεση) στον υπεύθυνο
- **γραπτές ασκήσεις** (Α, Β, Γ, ...)   
 παραδίδονται στον υπεύθυνο είτε χειρόγραφες είτε εκτυπωμένες (από υπολογιστή ή γραφομηχανή)
- **Θα δίνονται οδηγίες για τη σωστή λύση τους**
- **Κάθε άσκηση έχει τελική ημερομηνία,** μετά την οποία δε γίνεται δεκτή η παράδοση

## ◆ 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

## ◆ Τεχνητή γλώσσα

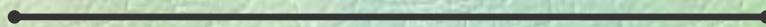
- Αυστηρότατη σύνταξη και σημασιολογία

## ◆ Γλώσσα προγραμματισμού

- Τεχνητή 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 ⇒ διαφορικές εξισώσεις
- ~1940, **Zuse**, ηλεκτρονική (ψηφιακή) υπολογιστική μηχανή  
⇒ πρόγραμμα και δεδομένα, χωριστά
- 1945–1950, μοντέλο **von Neumann**  
⇒ πρόγραμμα και δεδομένα, από κοινού
- 1950–σήμερα, ραγδαία ανάπτυξη της τεχνολογίας των **ηλεκτρονικών υπολογιστών**

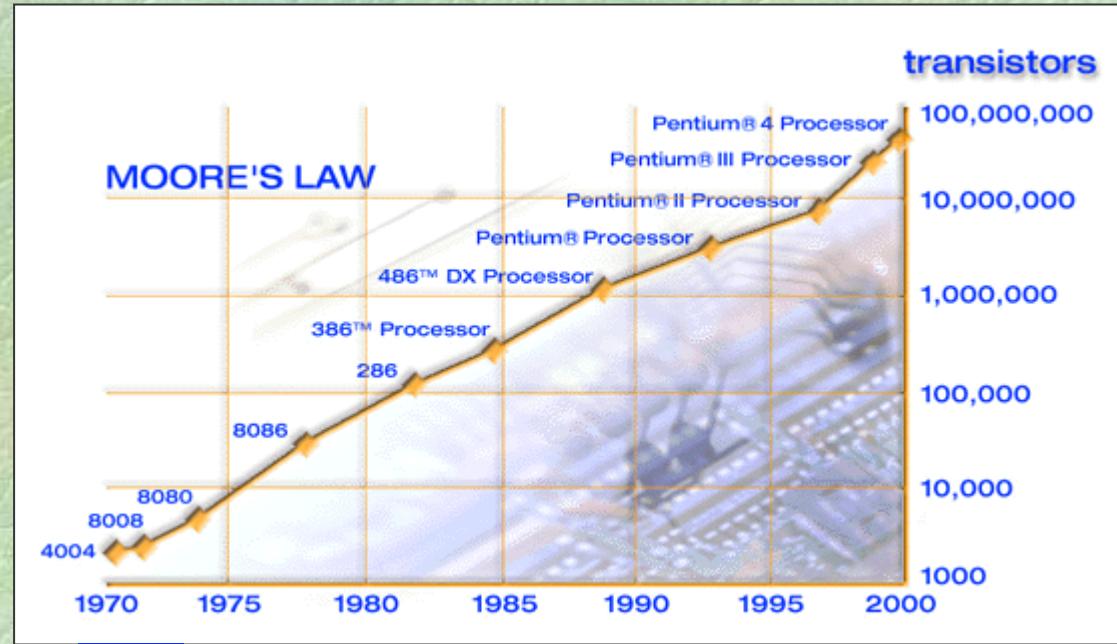
## ◆ Κατασκευή υπολογιστών

- 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

## ◆ 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*



## ◆ 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)

$A \wedge \neg A$  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

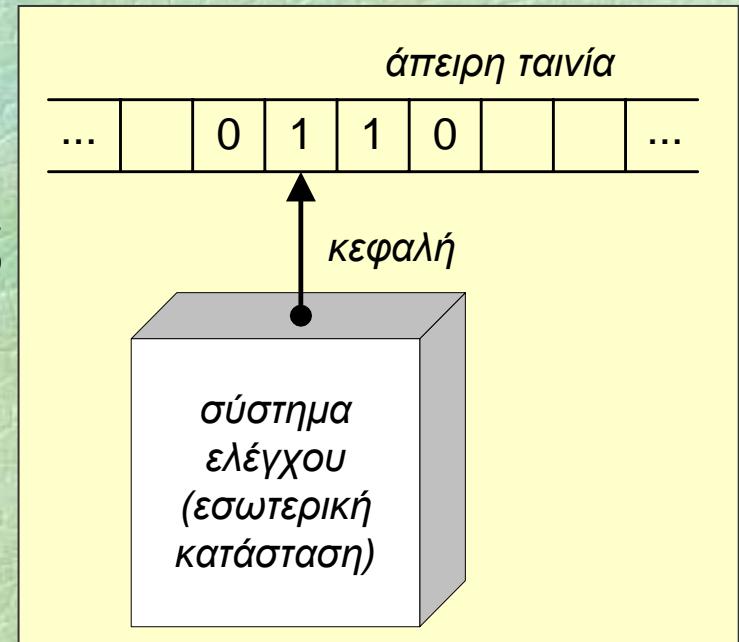
$$A = \{ x \mid x \notin x \}$$

$$\begin{array}{l} A \in A \rightarrow A \notin A \\ A \notin A \rightarrow A \in A \end{array}$$

- 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  
(θέατρο «Επί Κολωνώ»)
  - 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

διεύθυνση

11011011

εντολή

## ◆ Συμβολική γλώσσα (assembly)

label:

διεύθυνση

add

ax, bx

πράξη

δεδομένα

## ◆ Γλώσσες χαμηλού και υψηλού επιπέδου

## ◆ Υλοποίηση γλωσσών προγραμματισμού

- μεταγλωττιστής (compiler)
- διερμηνέας (interpreter)

## ◆ Κυριότερες γλώσσες, ιστορικά

- FORTRAN, Algol, LISP, COBOL, BASIC, PL/I
- Pascal
- Prolog, C, Smalltalk, Modula-2, Ada, C++, Java

## ◆ Pascal

- Niklaus Wirth (1971)
- Γλώσσα γενικού σκοπού (general purpose)
- Ευνοεί το συστηματικό και δομημένο προγραμματισμό

## ◆ Παραλλαγές

- Standard, ISO Pascal
- UCSD Pascal
- ANSI Pascal
- Turbo Pascal

# Ασκήσεις

(i)

```
program Hello1(output);
begin
    writeln('Hello, world')
end.
```

```
program Hello2(output);
begin
    writeln('Hello, ', 'world')
end.
```

```
program Hello3(output);
begin
    write('Hello, ') ; writeln('world')
end.
```

```
program Hello4(output);
begin
    write('Hello, world') ; writeln
end.
```

# Ασκήσεις

(ii)

```
program Hello5(output);  
  procedure hello;  
  begin  
    writeln('Hello, world')  
  end;  
  
begin  
  hello; hello  
end.
```

```
program Hello6(output);  
  var i : integer;  
  procedure hello;  
  begin  
    writeln('Hello, world')  
  end;  
  
begin  
  for i:=1 to 20 do hello  
end.
```

```
program Hello7(output);
  const n = 20;
  var i : integer;
  procedure num_hello;
begin
  writeln(i, ' Hello, world')
end;

begin
  for i:= 1 to n do num_hello
end.
```

```
program Hello8(input,output);
var i,n : integer;
p program Hello9(input,output);
b   var i,n : integer;
e   procedure hello;
begin
begin           writeln('Hello, world')
w   end;
b   begin
r   writeln('Give number of greetings',
f           'then press <enter>:');
end.
      readln(n);
      for i:= 1 to n do hello
end.
```

```
program Hello10(input,output);
  var i,n : integer;
  procedure hello;
  begin
    writeln('Hello, world')
  end;
begin
  writeln('Give number of greetings',
         'then press <enter>:');
  readln(n);
  if n < 0 then writeln('# is negative')
  else for i:= 1 to n do hello
end.
```

# Δομή του προγράμματος

(i)

```
program example(input, output);
```

επικεφαλίδα

```
var i, j : integer;
```

δηλώσεις

```
begin
```

```
    i:=15; j:=23;
```

```
    write('sum of i and j is: ');
```

```
    i:=i+j;
```

```
    write(i)
```

```
end.
```

κυρίως σώμα

# Δομή του προγράμματος

(ii)

## ◆ Επικεφαλίδα

header

program

identifier

(input, output)

τέλος

αρχή

βέλη: περιγράφουν  
τη σειρά με την οποία  
γράφονται τα κουτιά

το όνομα του τμήματος  
προγράμματος που ορίζεται

τετράγωνα κουτιά:  
απαιτούν εξήγηση

οβάλ κουτιά:  
γράφονται  
ακριβώς έτσι

## Συντακτικό διάγραμμα

- περιγράφει τη σύνταξη ενός τμήματος του προγράμματος

## ◆ Δηλώσεις μεταβλητών

- **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 δυνατών τιμών που μπορεί η μεταβλητή να πάρει
- var i : integer;**

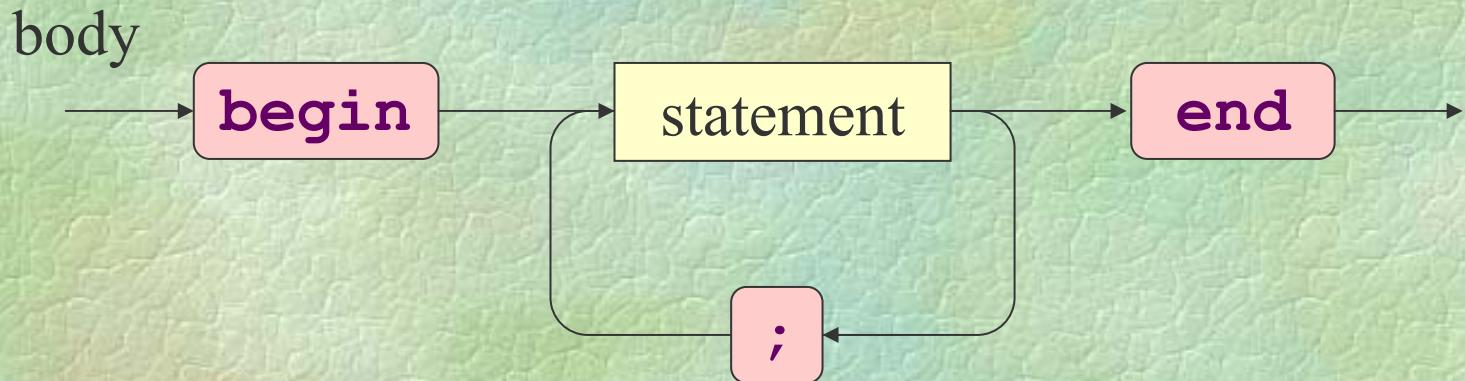
## ◆ Απλοί τύποι μεταβλητών

- **integer** ακέραιοι αριθμοί 0, 1, -3
- **real** πραγματικοί αριθμοί 3.14
- **char** χαρακτήρες 'a'
- **boolean** λογικές τιμές true, false

## ◆ Δήλωση περισσότερων μεταβλητών

```
var i,j,k : integer;  
x,y : real;      ch : char;  
changed : boolean;
```

## ◆ Κυρίως σώμα



## ◆ Σχόλια

```
var x,y : real; (* οι συντεταγμένες  
του κέντρου *)  
r : real; (* η ακτίνα *)
```

## ◆ Υποπρογράμματα

```
program example(input, output); επικεφαλίδα
```

```
    var i, j : integer;  
  
    procedure add;      (* ορισμός της *)  
    begin                (* διαδικασίας *)  
        i:=i+j  
    end;                  δηλώσεις
```

```
begin  
    i:=15; j:=23;  
    write('sum of i and j is: ' );  
  
    add;          (* κλήση της διαδικασίας *)  
    write(i)  
end.                  κυρίως σώμα
```

# Τί σημαίνει ορθό πρόγραμμα (i)

## ◆ Συντακτική ορθότητα

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

## ◆ Συντακτικά σφάλματα στην Pascal

- εμφανίζονται όταν δεν ικανοποιούνται τα συντακτικά διαγράμματα
- παράδειγμα:  
**( program ) input example, output;**

## ◆ Νοηματική ορθότητα

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

## ◆ Νοηματικά σφάλματα στην Pascal

- εσφαλμένη χρήση τελεστών  
`n := 'a' + 1`
- χρήση μεταβλητών χωρίς δήλωση  
`var n,i : integer;`  
`begin`  
 `n := i + j`

# Τί σημαίνει ορθό πρόγραμμα (iii)

## ◆ Σημασιολογική ορθότητα

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

## ◆ Σημασιολογικά σφάλματα στην Pascal

- 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`

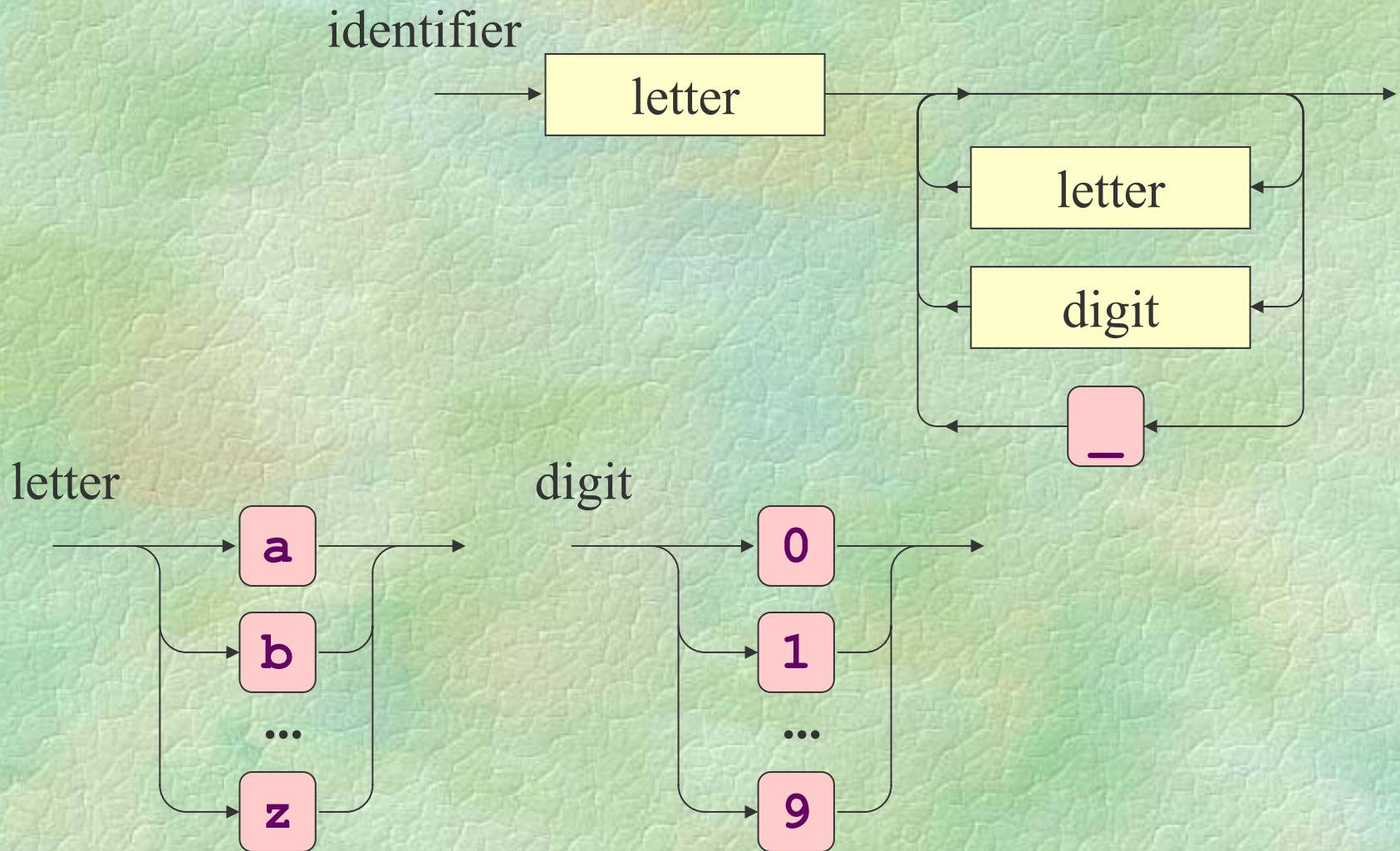
διαιρεση με  
το μηδέν

# Τί 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



# 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)
```

# Επικοινωνία με το χρήστη (i)

## ◆ Έξοδος στην οθόνη

```
write('Hello world')
```

```
write(x)
```

```
write(n+1)
```

```
write(x, y)
```

```
write('Η τιμή του x είναι ', x)
```

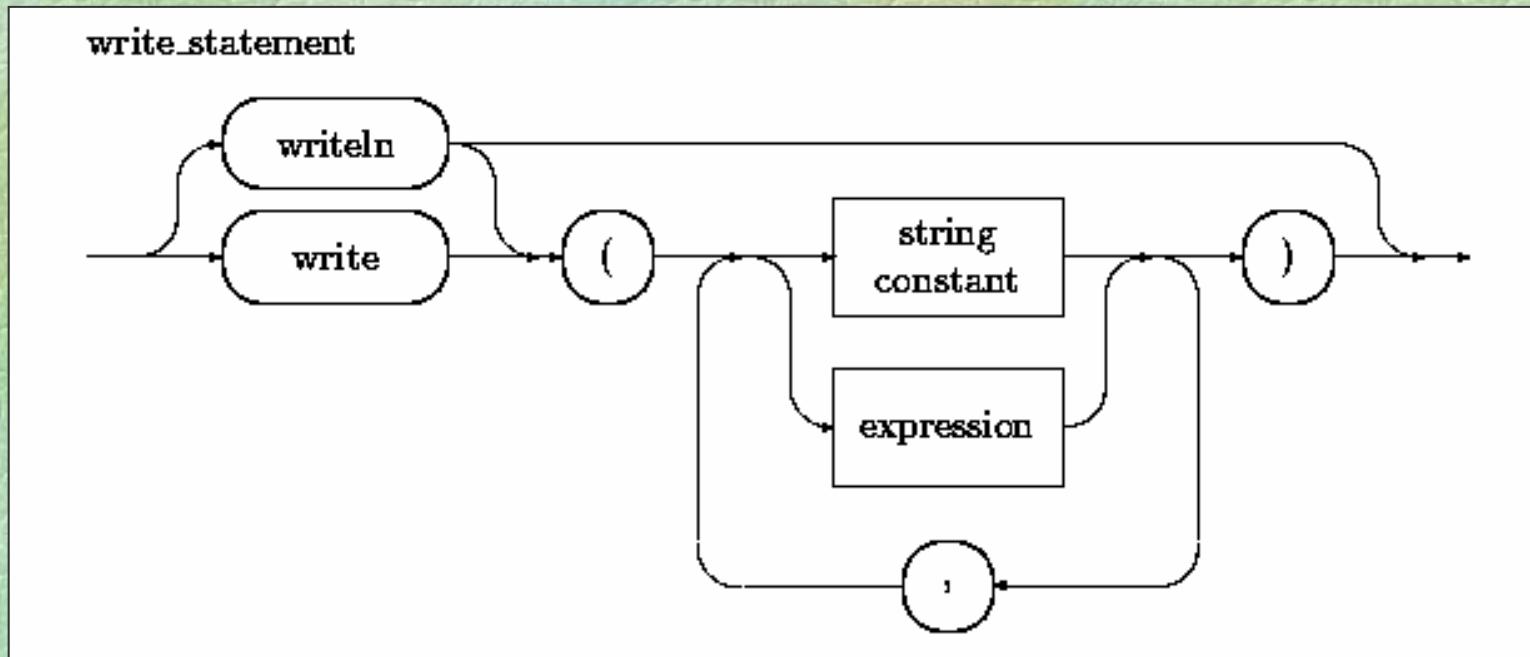
## ◆ Έξοδος με αλλαγή γραμμής

```
write('Η τιμή του');
```

```
writeln(' x είναι ', x);
```

```
writeln('Η τιμή του y είναι ', y)
```

## ◆ Συντακτικό διάγραμμα



- ◆ Είσοδος από το πληκτρολόγιο  
`read(a)`
- ◆ Είσοδος από το πληκτρολόγιο και διάβασμα μέχρι το τέλος της γραμμής  
`readln(b)`  
`readln(x, y)`

### ◆ Παράδειγμα

```
program example(input,output);
  var n, m, sum : integer;
begin
  writeln('Προσθέτω δυο ακέραιους');
  write('Δώσε το n: ');
  readln(n);
  write('Δώσε το m: ');
  readln(m);
  sum := n + m;
  write('Το άθροισμα ', n, ' + ', m,
        ' είναι: ');
  writeln(sum)
end.
```

## ◆ Απλές παραστάσεις

- σταθερές και μεταβλητές

## ◆ Απλές πράξεις

- πρόσθεση, αφαίρεση +, -
- πολλαπλασιασμός \*
- διαίρεση πραγματικών αριθμών /
- πηλίκο ακέραιας διαίρεσης **div**
- υπόλοιπο ακέραιας διαίρεσης **mod**
- πρόσημα +, -

## ◆ 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 (και) and
- διάζευξη (ή) or
- άρνηση (όχι) not

## ◆ Πίνακες αλήθειας λογικών πράξεων

<b>p</b>	<b>q</b>	<b>p and 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>

<b>p</b>	<b>q</b>	<b>p or 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>

<b>p</b>	<b>not p</b>
<b>false</b>	<b>true</b>
<b>true</b>	<b>false</b>

## ◆ Προτεραιότητα λογικών τελεστών

- **not** : μεγαλύτερη προτεραιότητα από όλους
- **and** : όπως ο πολλαπλασιασμός
- **or** : όπως η πρόσθεση
- π.χ. **not p and q or r**  
 $\equiv ((\text{not } p) \text{ and } q) \text{ or } r$
- π.χ. **x>3 and not y=5** **Λάθος!**  
 $\equiv x > (3 \text{ and } (\text{not } y)) = 5$
- π.χ. **(x>3) and not (y=5)** **Σωστό**

# Δομές ελέγχου

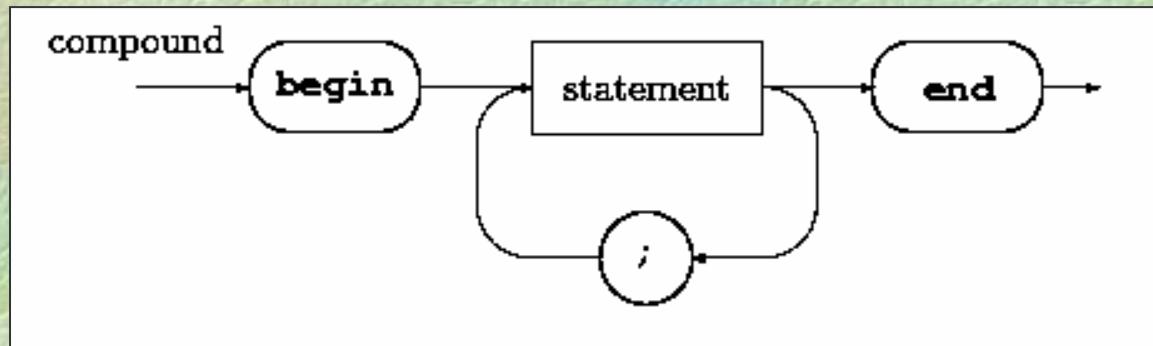
---

- ◆ 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

# Σύνθετη εντολή

(i)

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



## ◆ Παραδείγματα

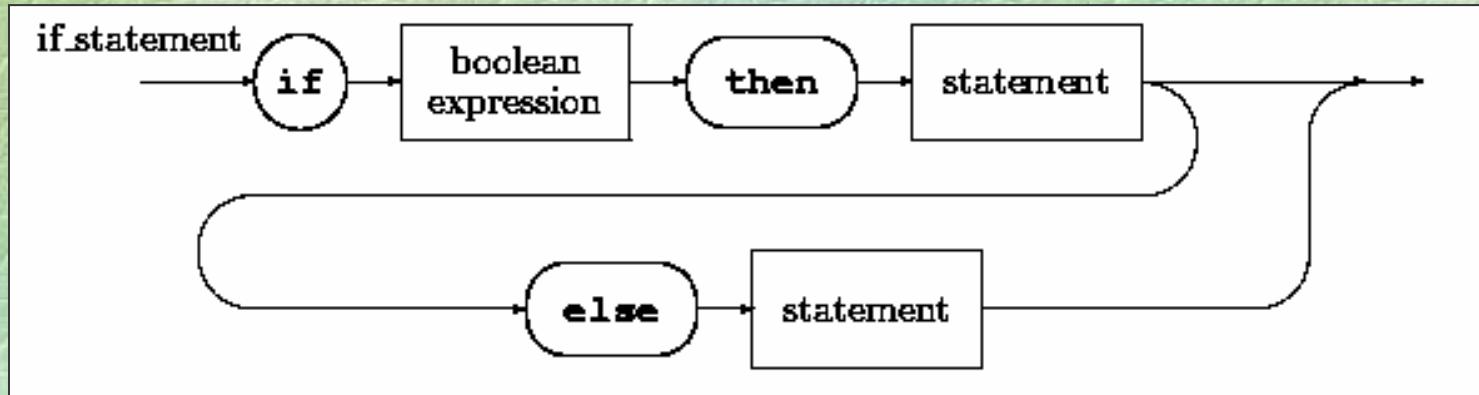
```
begin
    x:= 2; y:=3; z:=3;
    writeln(x, y, z)
end
```

```
begin
    x:= 2; y:=3;
    begin
        z:=3;
        write(x, y, z)
    end;
    writeln
end
```

# Εντολή if

(i)

- ◆ Εκτέλεση εντολών υπό συνθήκη
- ◆ Συντακτικό διάγραμμα



# Εντολή if

---

(ii)

## ◆ Παραδείγματα

```
if x>10 then x:=x+1
```

```
if age<10 then write('παιδί')
```

```
if (year>1900) and (year<=2000) then  
    write('20ός αιώνας')
```

```
if (year mod 4 = 0) and  
    (year mod 100 <> 0) or  
    (year mod 400 = 0) and  
    (year mod 4000 <> 0) then  
    write('δίσεκτο έτος')
```

# Εντολή if (iii)

## ◆ Παραδείγματα (συνέχεια)

```
if changed then
begin
  writeln('Το αρχείο άλλαξε');
  changed := false
end

if x mod 2 = 0 then write('άρτιος')
else write('περιττός')

if mine then begin me:=1; you:=0 end
else begin me:=0; you:=1 end

if x > y then write('μεγαλύτερο')
else if x < y then write('μικρότερο')
else write('ίσο')
```

# Εντολή if (iv)

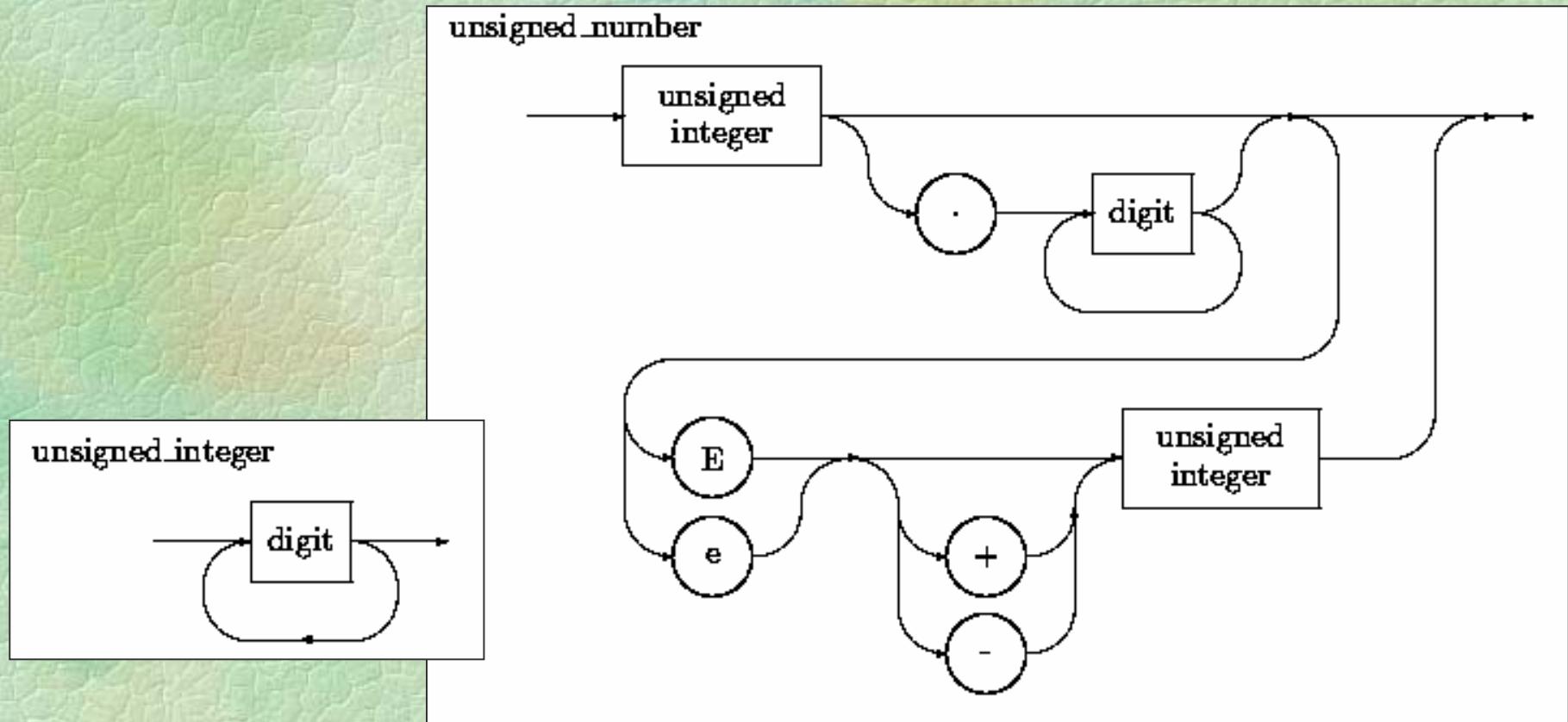
- ◆ Ένα **else** αντιστοιχεί στο πλησιέστερο προηγούμενο **if** που δεν έχει ήδη αντιστοιχιστεί σε άλλο **else**
- ◆ Παράδειγμα

```
if x>0 then
    if y>0 then
        write('πρώτο τεταρτημόριο')
    else if y<0 then
        write('τέταρτο τεταρτημόριο')
    else
        write('άξιονας των x')
```

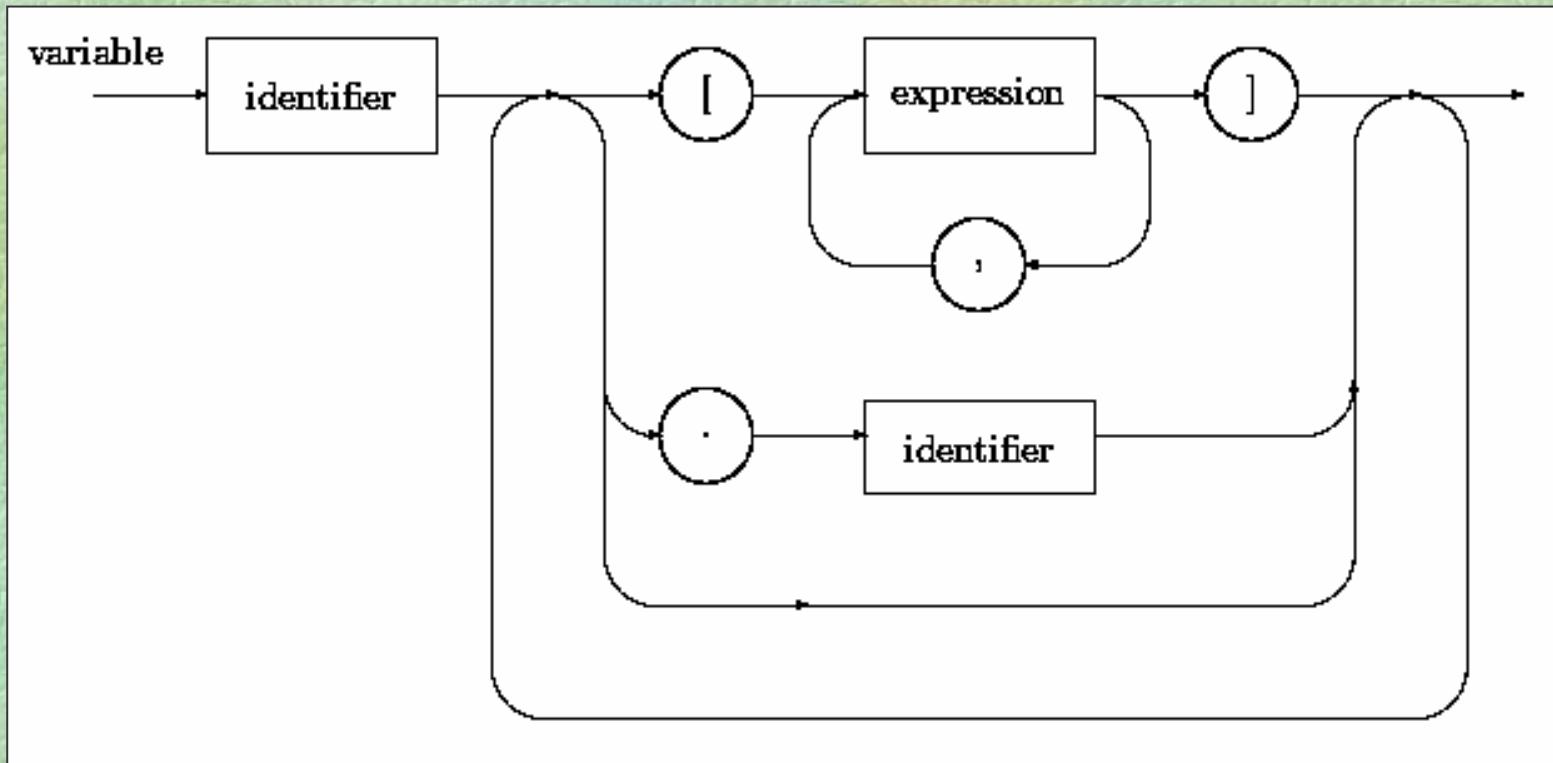
# Σύνταξη παραστάσεων

(i)

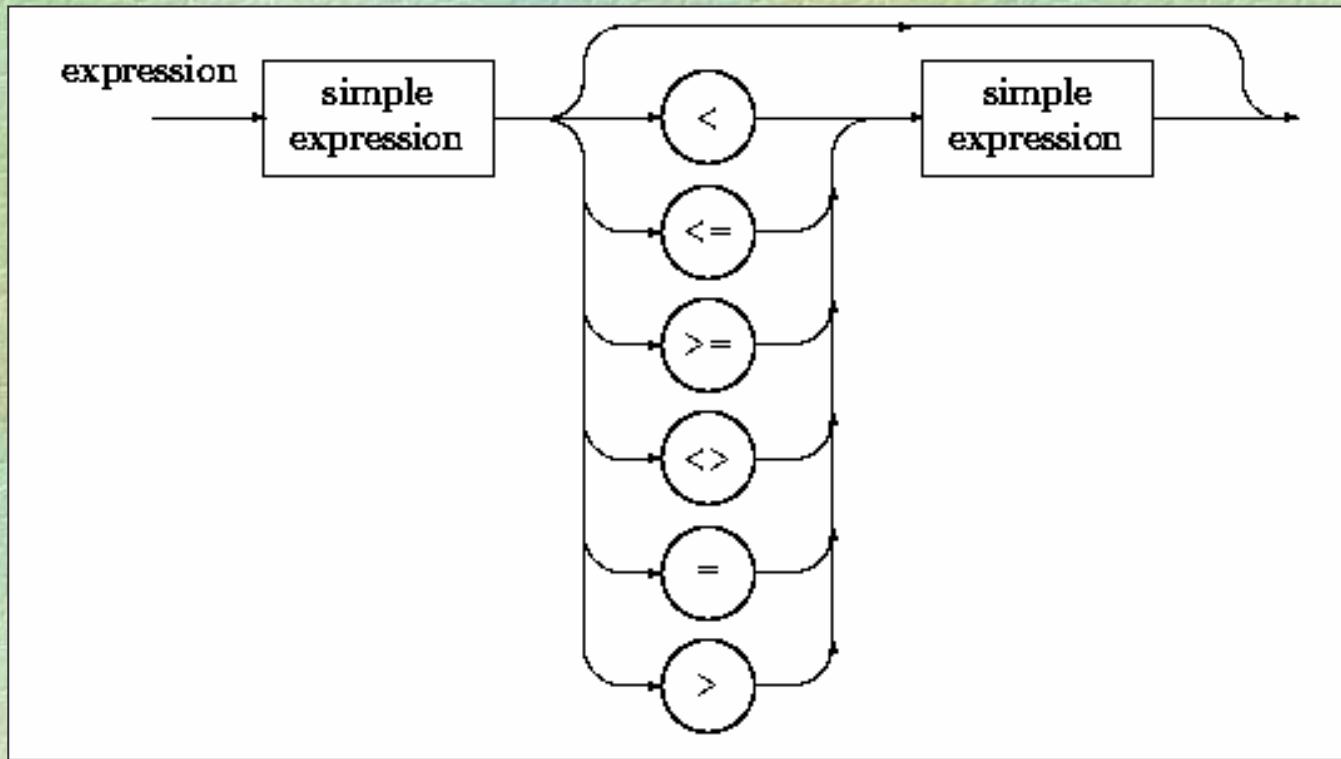
## ◆ Σταθερές



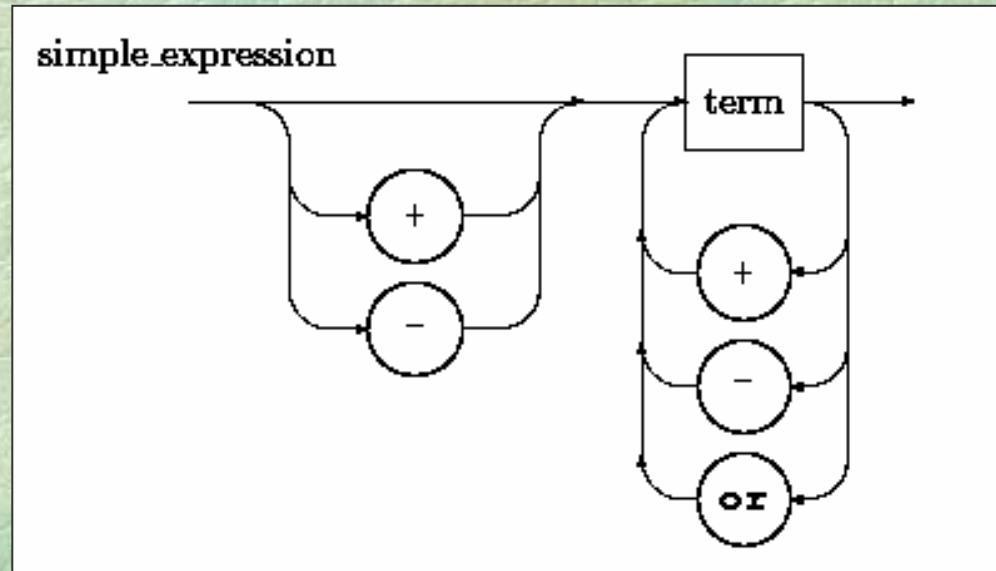
## ◆ Μεταβλητές



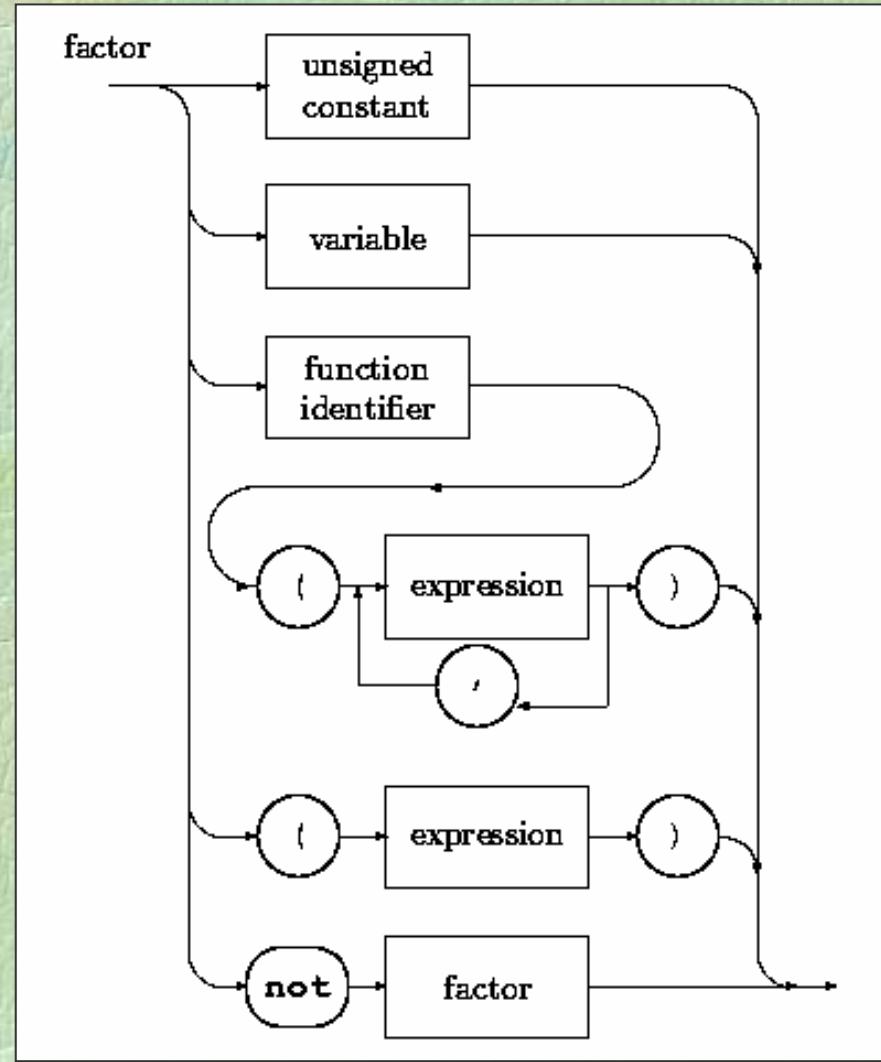
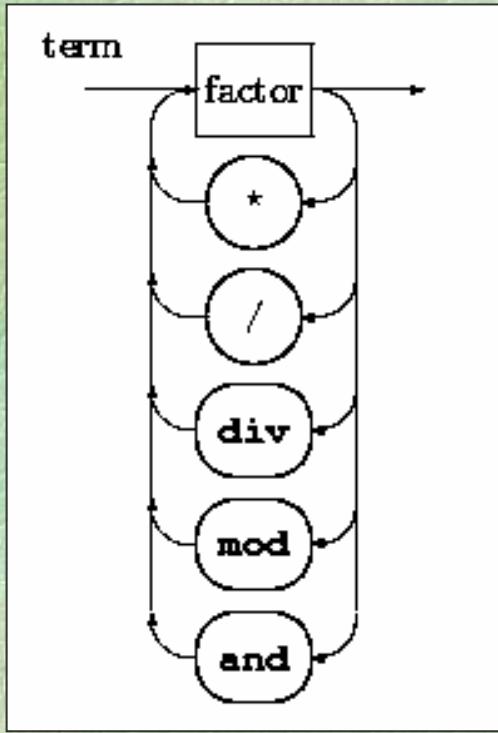
## ◆ Παράσταση



## ◆ Απλή παράσταση



## ◆ Όροι και παράγοντες



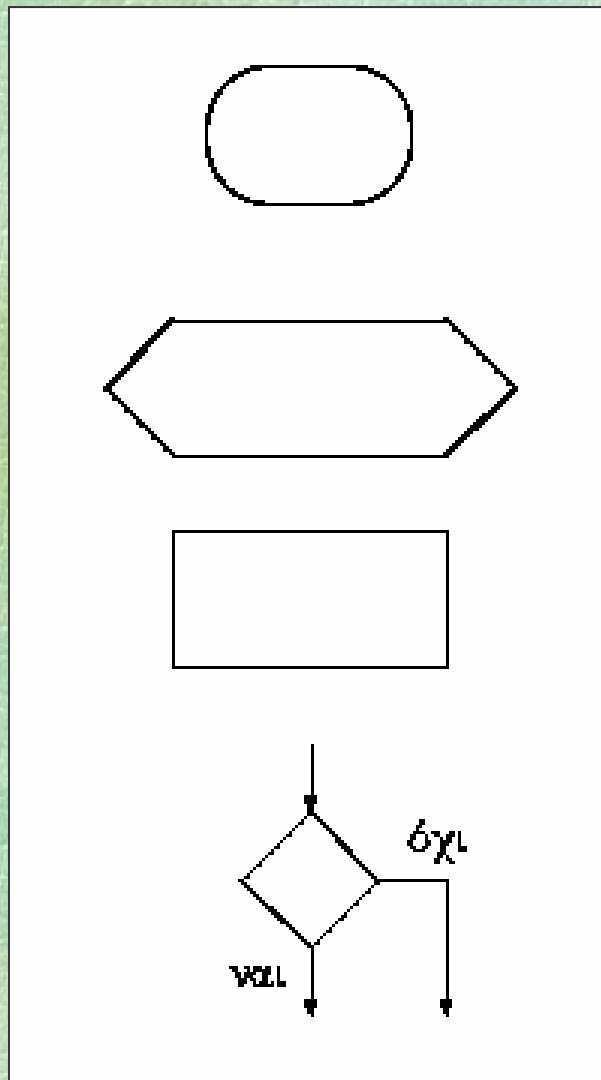
# Παραδείγματα παραστάσεων (i)

Συντακτική ορθή παράσταση	Πώς διαβάζεται (σχόλια)	Τύπος αποτελέσματος	Τιμή αποτελέσματος
<code>5 = 1</code>	5 ισον με 1	boolean	false
<code>5 &gt;= 1</code>	5 μεγαλύτερο ή ίσον με το 1	boolean	true
<code>5 &lt;&gt; 1</code>	5 διάφορο από το 1	boolean	true
<code>grade &gt; small</code>	grade μεγαλύτερο του small	boolean	?
<code>not done</code>	άρνηση του done	boolean	?
<code>(5 &lt;&gt; 1) and (1 &gt; 2)</code>	Χρήση του τελεστή and	boolean	false

# Παραδείγματα παραστάσεων (ii)

Συντακτική ορθή παράσταση	Πός διαβάζεται (σχόλια)	Τύπος αποτελέσματος	Τιμή αποτελέσματος
(5 <> 1) <b>or</b> (2 < 1)	Χρήση του τελεστή <b>or</b>	boolean	true
(6 <b>div</b> 3) > 1	Χρήση του τελεστή <b>div</b>	boolean	true
(t > 1) <b>or</b> (k > 2)	Χρήση του τελεστή <b>or</b>	boolean	?
(7 <b>mod</b> 2 > 0) <b>and</b> (10 <b>div</b> 2 > 3)	Χρήση διαφόρων πράξεων	boolean	true
9 * (10E-2) *101		real	90.9
10E3 - 9 * 3 * (24 <b>mod</b> 6 + 1)		real	9973.0

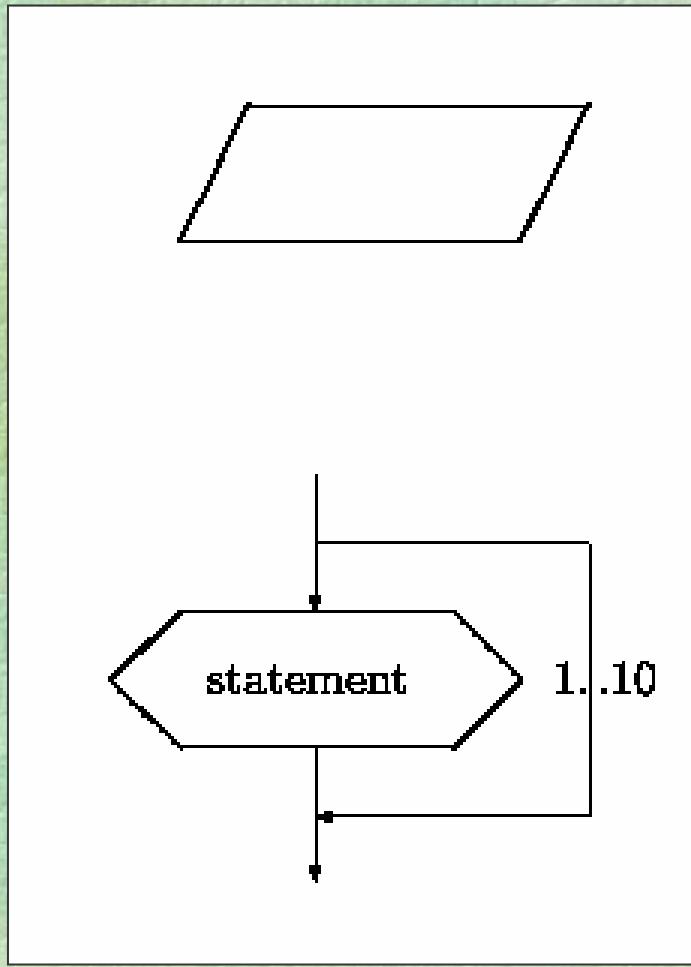
# Λογικά διαγράμματα ροής (i)



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

# Λογικά διαγράμματα ροής

(ii)



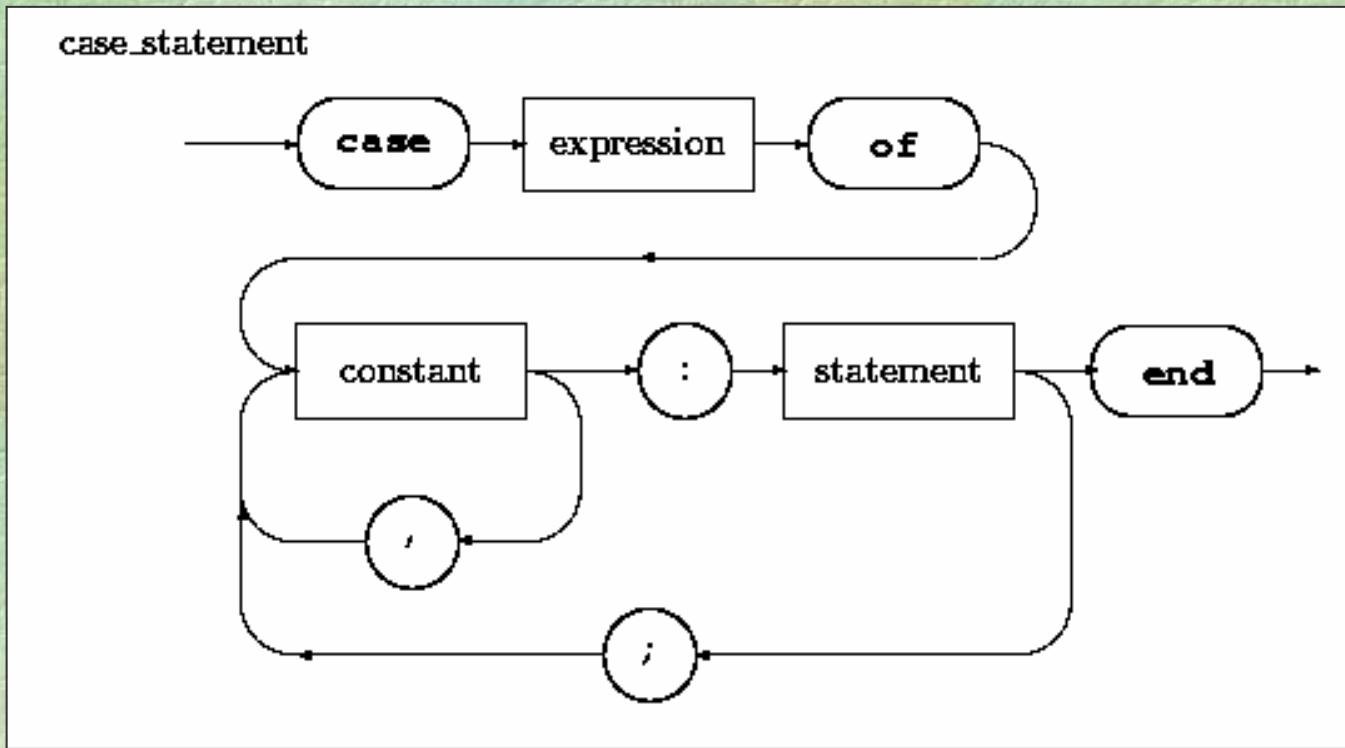
◆ Λειτουργία εισόδου/εξόδου

◆ Επανάληψη (βρόχος)

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

```
if month=1 then
    write('Ιανουάριος')
else if month=2 then
    write('Φεβρουάριος')
else if month=3 then
    write('Μάρτιος')
else if ...
    ...
else if month=12 then
    write('Δεκέμβριος')
```

## ◆ Συντακτικό διάγραμμα



## ◆ Παραδείγματα

```
case month of
```

```
 1: write('Ιανουάριος');  
 2: write('Φεβρουάριος');  
 3: write('Μάρτιος');  
 ...  
 12: write('Δεκέμβριος')
```

```
end (* case *)
```

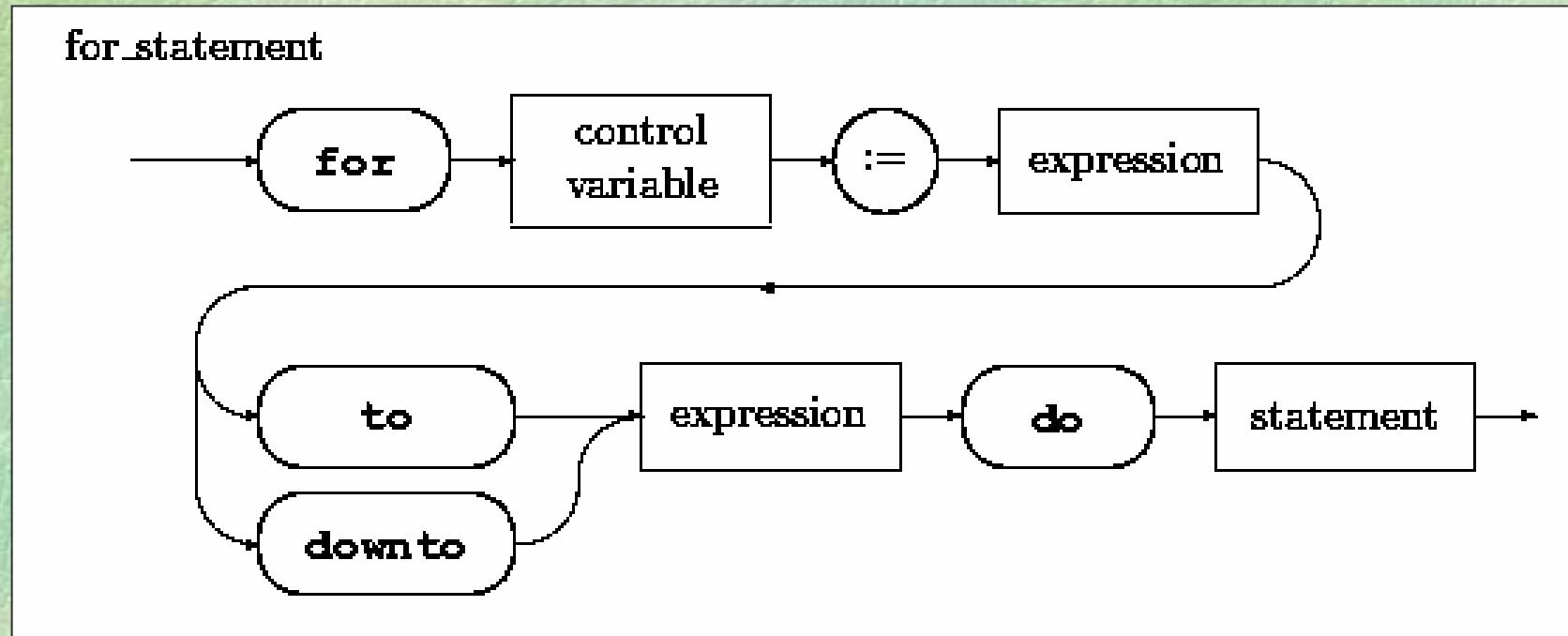
```
case month of
```

```
 1,3,5,7,8,10,12 : write('31 μέρες');  
 4,6,9,11          : write('30 μέρες');  
 2                 : write('28 ή 29')  
end (* case *)
```

# Εντολή for

(i)

- ◆ Βρόχος με σταθερό αριθμό επαναλήψεων
- ◆ Συντακτικό διάγραμμα



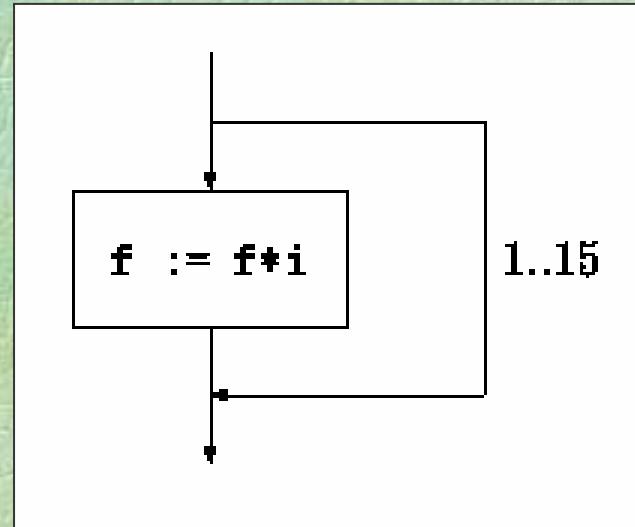
## ◆ Παραδείγματα

```
for i:=1 to 10 do writeln(i)
```

```
for i:=10 downto 1 do writeln(i)
```

```
for i:=41 downto -3 do  
  write('*')
```

```
f:=1;  
for i:=1 to 15 do  
  f:=f*i
```



## ◆ Παραδείγματα (συνέχεια)

```
for i:=1 to 5 do
begin
    for j:=1 to 10 do
        write('*');
    writeln
end

for i:=1 to 5 do
begin
    for j:=1 to 2*i do
        write('*');
    writeln
end
```

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

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

- ◆ Ειδικές περιπτώσεις για τα όρια:

**for i:=10 to 10 do ... (\* μία φορά \*)**

**for i:=12 to 10 do ... (\* καμία φορά \*)**

- ◆ Η μεταβλητή ελέγχου δεν είναι ορισμένη μετά το τέλος του βρόχου

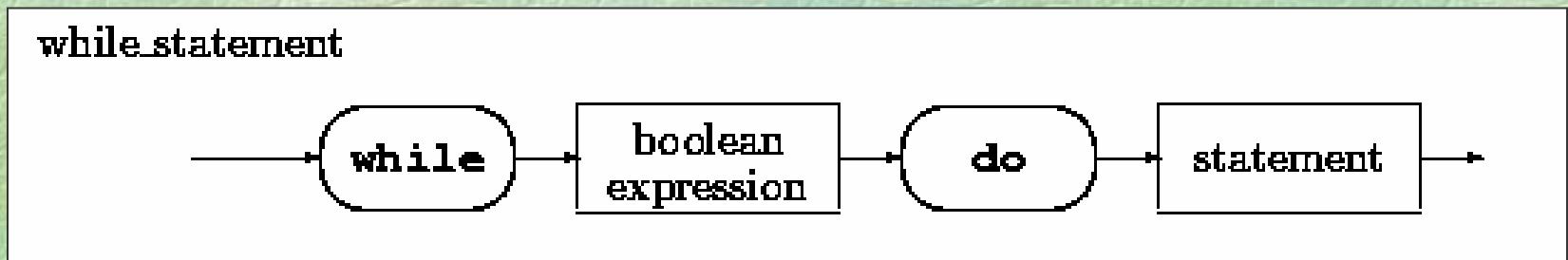
- ◆ Η μεταβλητή ελέγχου δεν μπορεί να μεταβληθεί (π.χ. με ανάθεση) μέσα στο σώμα του βρόχου

- ◆ Τα όρια υπολογίζονται μια φορά στην αρχή

# Εντολή while

(i)

- ◆ Βρόχος όσο ικανοποιείται μια συνθήκη
- ◆ Συντακτικό διάγραμμα



- ◆ Παραδείγματα

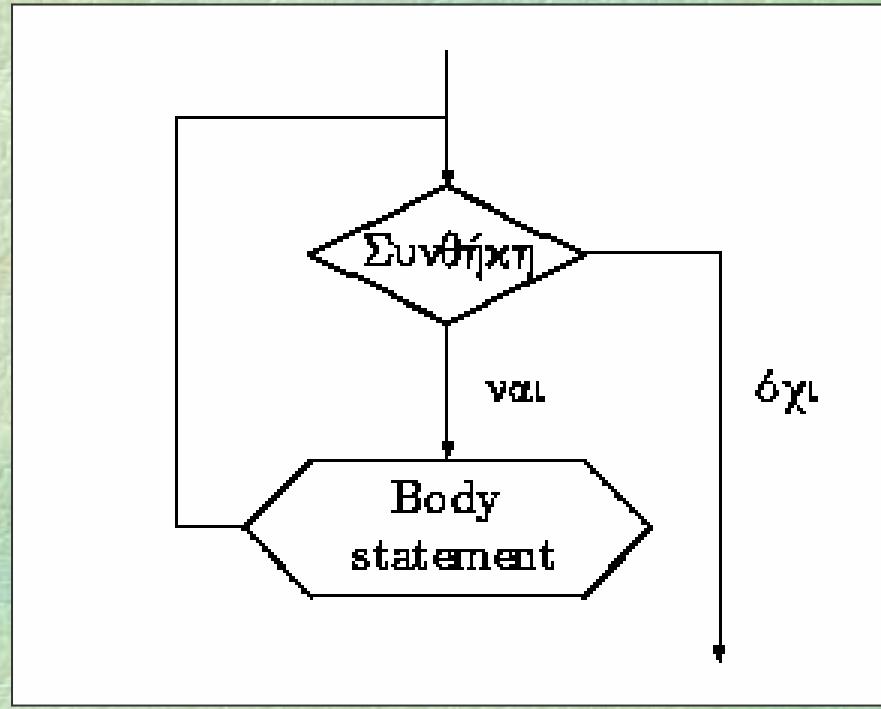
**while** x>15 **do** k:=k+2

**while** state **and** (x>15) **do**  
**begin** x:=x-5; **write**(x) **end**

# Εντολή while

(ii)

- ◆ Διάγραμμα ροής  
**while συνθήκη do σώμα**

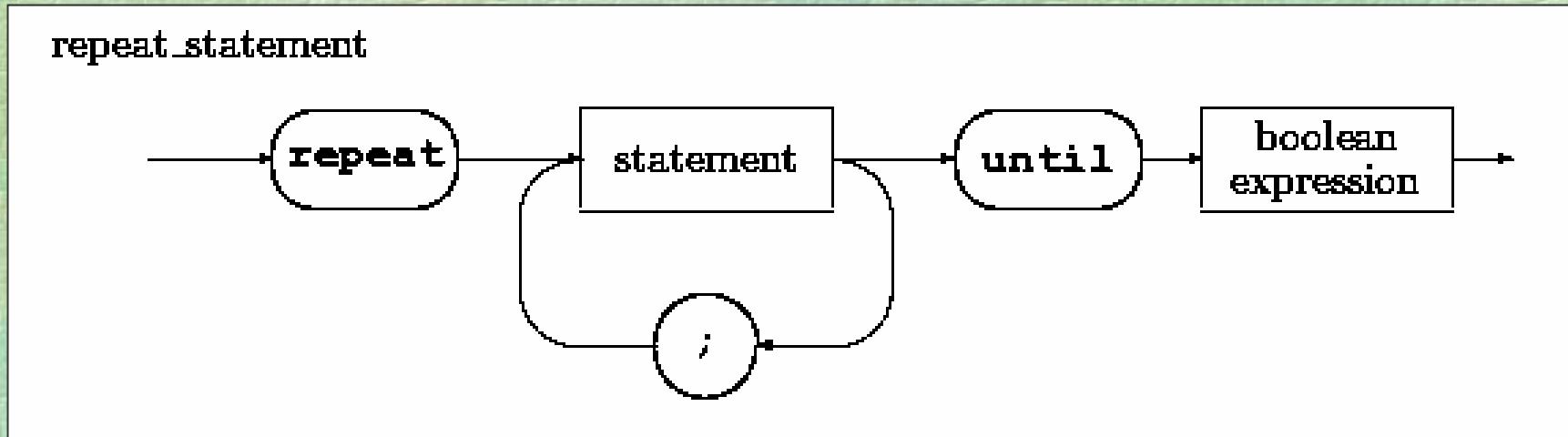


- ◆ Ο αριθμός επαναλήψεων γενικά δεν είναι γνωστός εκ των προτέρων
- ◆ Αν η συνθήκη είναι αρχικά ψευδής, ο βρόχος τερματίζεται χωρίς να εκτελεστεί το σώμα

# Εντολή repeat

(i)

- ◆ Βρόχος μέχρι να ικανοποιηθεί μια συνθήκη
- ◆ Συντακτικό διάγραμμα

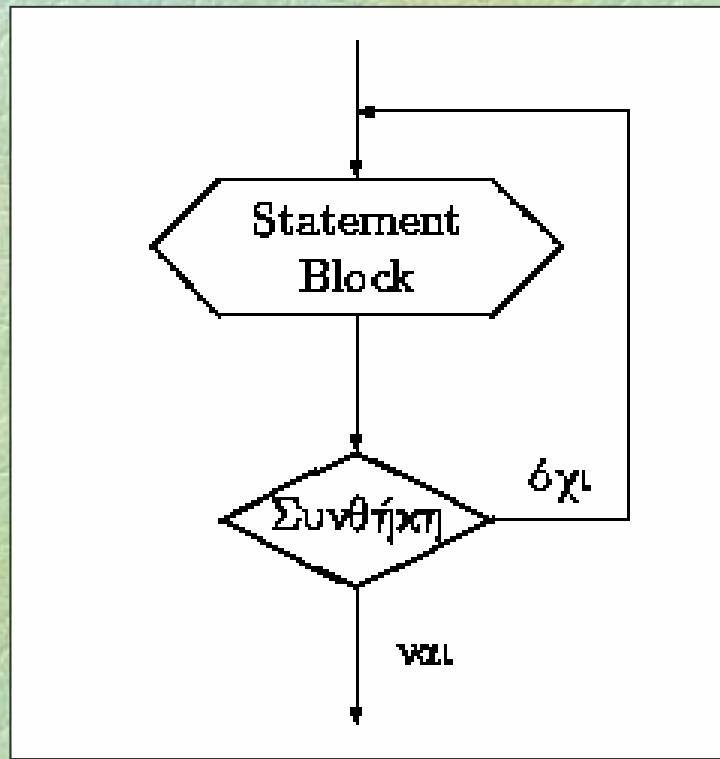


- ◆ Παράδειγμα

`x:=1;`

`repeat writeln(x); x:=2*x until x>=500`

## ◆ Διάγραμμα ροής



- ◆ 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

# Εντολή repeat

(iv)

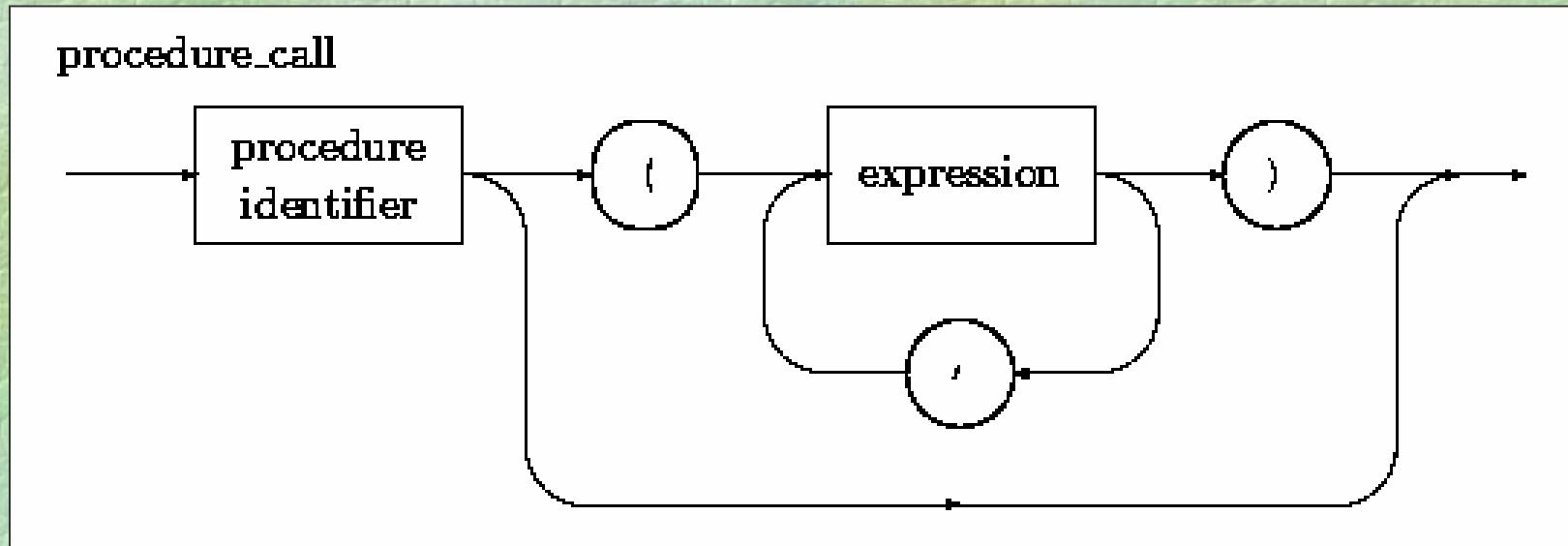
## ◆ Παράδειγμα

```
program primes(input, output);
  var p,t : integer;
begin
  writeln(2); p:=1;
  repeat p:=p+2; t:=1;
    repeat t:=t+2
    until p mod t = 0;
    if p=t then writeln(p)
  until p>100
end.
```

p	t
1	1
3	3
5	1
2	3
3	5
5	7
7	1
7	3
11	5
...	7
9	1
11	3
...	

# Κλήση διαδικασίας

## ◆ Συντακτικό διάγραμμα



## ◆ Παραδείγματα

`myproc(x, y+1, 3)`

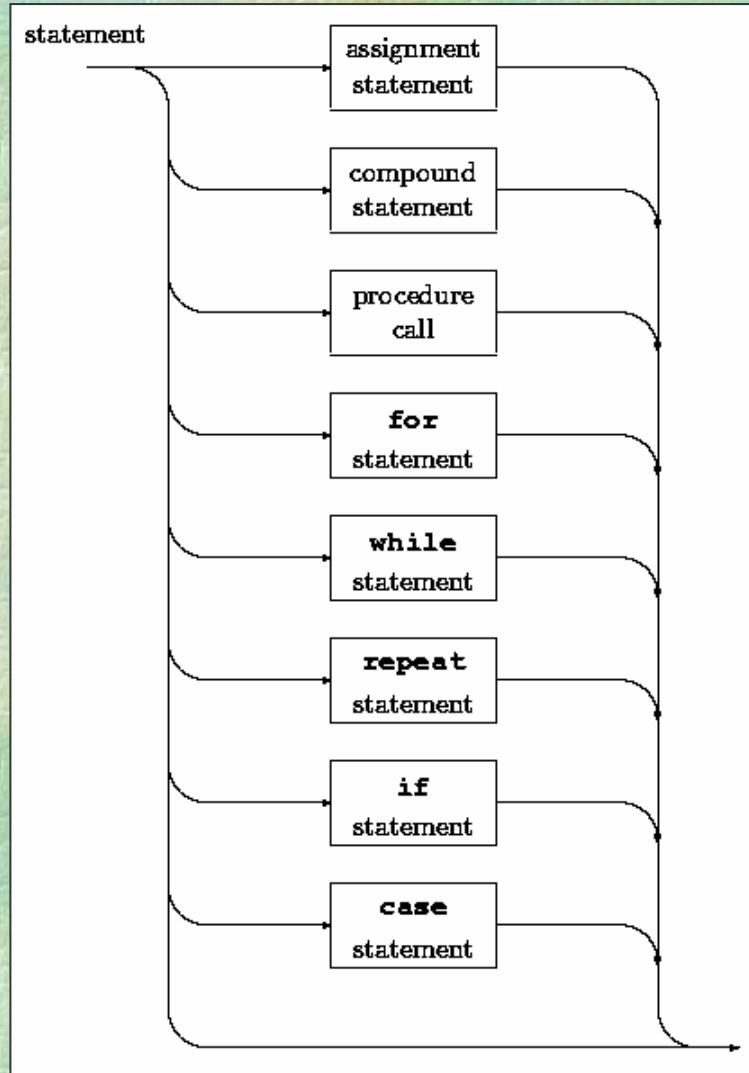
`write(x+y, 'wednesday', x-y); writeln`

# Κενή εντολή

- ◆ Συμβολίζεται με την κενή συμβολοσειρά
- ◆ Δεν κάνει τίποτα όταν εκτελείται
- ◆ Παράδειγμα

```
if x>4 then  
begin  
    y:=1;  
    x:=x-5;      (* εδώ υπάρχει μια  
                   κενή εντολή *)  
end
```

# Συντακτικό διάγραμμα εντολής



- ◆ **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

# Δομημένος προγραμματισμός (ii)

```
program tasks(input, output);
var n1, n2, sum, expr : integer;
procedure arith(x,y : integer;
                var s,e : integer);
begin s := x+y;
      e := sqr(x)+3*y+5
end;

begin
  write('Δώσε το n1: '); readln(n1);
  write('Δώσε το n2: '); readln(n2);
  arith(n1, n2, sum, expr);
  writeln('Αθροισμα = ', sum,
          ' έκφραση = ', expr)
end.
```

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

- ◆ 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 στο κυρίως πρόγραμμα

- ◆ Τυπικές (formal) παράμετροι ενός υποπρογράμματος είναι οι 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

- ◆ Παράμετροι τιμών (κλήση με τιμή):  
πέρασμα πληροφοριών από το καλούν  
(πρόγραμμα) προς το καλούμενο  
(υποπρόγραμμα)
- ◆ Παράμετροι μεταβλητών (κλήση με αναφορά)  
πέρασμα πληροφοριών από το καλούν  
(πρόγραμμα) προς το καλούμενο  
(υποπρόγραμμα)  
*και αντίστροφα*

```
program tasks2(input, output);
  var a, b : integer;
  procedure nochange(x : integer);
    var d : integer;
  begin write('nochange: ', );
    d:=2*x; x:=x+d; writeln(x)
  end;
  procedure change(var y : integer);
  begin write('change: ', );
    y:=y*2; writeln(y)
  end;
begin a:=1; b:=2; writeln(a,b);
  nochange(a); change(a); writeln(a,b);
  nochange(b); change(b); writeln(a,b)
end.
```

- ◆ Όπως οι διαδικασίες, αλλά επιστρέφουν μια τιμή ως **αποτέλεσμα**
- ◆ Δεν μπορούν να χρησιμοποιηθούν ως εντολές αλλά μόνο σε παραστάσεις

## ◆ Παράδειγμα

```
program funcall(input, output);
  var n1, n2, n3 : integer;
  function average(a,b,c : integer) :
    integer;
begin
  average := (a+b+c) div 3
end;

begin n1:=10; n2:=15; n3:=20;
  writeln('Average: ',
    average(n1, n2, n3))
end.
```

# Βαθμιαία 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

## ◆ Περιγραφή προβλήματος

- Δίνονται δύο θετικοί ακέραιοι  $a, b$
- Ζητείται ο μέγιστος κοινός διαιρέτης τους

## ◆ Απλός αλγόριθμος

```
z := min(a, b);  
while (a mod z <> 0)  
      or (b mod z <> 0) do z := z-1;  
writeln(z)
```

- Ο αριθμός επαναλήψεων του βρόχου είναι της τάξης του  $\min(a, b)$

## ◆ Αλγόριθμος με αφαιρέσεις

- Ιδέα: αν  $i > j$  τότε  $\text{gcd}(i, j) = \text{gcd}(i-j, j)$

- Αντίστοιχο πρόγραμμα

`i := a; j := b;`

`while (i>0) and (j>0) do`

`if i>j then i := i-j else j := j-i;`

`writeln(i+j)`

- Στη χειρότερη περίπτωση, ο αριθμός επαναλήψεων είναι της τάξης του  $\max(a, b)$
- Στη μέση περίπτωση όμως, αυτός ο αλγόριθμος είναι καλύτερος του προηγούμενου

## ◆ Αλγόριθμος του Ευκλείδη

- Ιδέα: αντί πολλαπλών αφαιρέσεων χρησιμοποιούμε το υπόλοιπο της διαίρεσης
- Αντίστοιχο πρόγραμμα

```
i := a; j := b;  
while (i>0) and (j>0) do  
    if i>j then i := i mod j  
    else j := j mod i;  
writeln(i+j)
```

- Στη χειρότερη περίπτωση, ο αριθμός επαναλήψεων είναι της τάξης του  $\log(a+b)$

```
program gcd(input, output);
δηλώσεις;
begin αρχικό μήνυμα στο χρήστη;
repeat μήνυμα που ζητάει είσοδο;
διάβασμα δύο ακεραίων;
if ορθή είσοδος then
begin αρχικοποίηση βρόχου;
while δεν τελείωσε do
χρήση της ιδέας του Ευκλείδη;
παρουσίαση αποτελεσμάτων
end
else μήνυμα σφάλματος
until συμφωνημένη είσοδος για τέλος
end.
```

Πρώτη  
προσέγγιση

## ◆ Συγκεκριμενοποίηση

- χρήση της ιδέας του Ευκλείδη

```
if i>j then i := i mod j  
else j := j mod i
```

- συνθήκη «δεν τελείωσε»

$(i > 0) \text{ and } (j > 0)$

- αρχικοποίηση βρόχου

$i := a; j := b$

- διάβασμα δύο ακεραίων

**readln(a, b)**

Δεύτερη  
προσέγγιση

## ◆ Συγκεκριμενοποίηση (συνέχεια)

- ορθή είσοδος  
 $(a>0)$  **and**  $(b>0)$
- συμφωνημένη είσοδος για τέλος  
 $(a=0)$  **and**  $(b=0)$
- απομένουν μόνο: δηλώσεις, μηνύματα και σχόλια

Δεύτερη προσέγγιση

## ◆ Συμπληρώνουμε τα παραπάνω στο πρόγραμμα

## ◆ Το τελικό πρόγραμμα

(\* Ευκλείδιος αλγόριθμος  
Στάθης Ζάχος, 19 Οκτωβρίου 1990 \*)

```
program gcd(input, output);  
  
var a,b,i,j : integer;  
  
begin  
    writeln('Θα υπολογίσω το μ.κ.δ. ',  
           'δύο θετικών ακεραίων');  
repeat  
    writeln('Δώσε δύο θετικούς ',  
           'ακέραιους ή 0 0 για έξοδο');  
    readln(a, b);
```

◆ Το τελικό πρόγραμμα (συνέχεια)

```
if (a>0) and (b>0) then
begin i:=a; j:=b;
  while (i>0) and (j>0) do
    if i>j then i := i mod j
    else j := j mod i;
  writeln('μκδ(', a, ',', b,
         ') = ', i+j)
end
  else writeln('δεν είναι θετικοί')
until (a=0) and (b=0)
end.
```

# Αθροιστικό πρόγραμμα

(i)

```
program addition(input, output);  
  δηλώσεις;  
begin αρχικό μήνυμα στο χρήστη;  
  repeat  
    μήνυμα που ζητάει είσοδο;  
    áθροιση αριθμών;  
    παρουσίαση αποτελεσμάτων;  
    μήνυμα για να ξέρει ο χρήστης  
      πώς θα συνεχίσει ή θα σταματήσει  
  until συμφωνημένη είσοδος για τέλος  
end.
```

Πρώτη  
προσέγγιση

## ◆ Συγκεκριμενοποίηση

- άθροιση αριθμών  
*αρχικοποίηση βρόχου ;  
repeat*  
*είσοδος ενός αριθμού ;  
άθροιση ;  
είσοδος ενός συμβόλου  
**until** σύμβολο διάφορο του +*

Δεύτερη  
προσέγγιση

## ◆ Περαιτέρω συγκεκριμενοποίηση

- άθροιση αριθμών

```
sum := 0;  
repeat  
    read(number);  
    sum := sum + number;  
    read(symbol)  
until symbol <> '+'
```

Τρίτη  
προσέγγιση

## ◆ Το τελικό πρόγραμμα

(\* Αθροιστικό πρόγραμμα  
Στάθης Ζάχος, 19 Οκτωβρίου 1990 \*)

```
program addition(input, output);

var number,sum : integer;
    symbol      : char;

begin
    writeln('Πρόσθεση ακεραίων');
    repeat
        writeln('Δώσε τους ακεραίους ',
               'χωρισμένους με + ',
               'ή δώσε = για τέλος');


```

## ◆ Το τελικό πρόγραμμα (συνέχεια)

```
sum := 0;
repeat read(number);
    sum := sum + number;
    read(symbol)
until symbol <> '+';
if symbol <> '=' then write('=');
writeln(sum);
write('Θέλεις να συνεχίσεις; ',
      'Δώσε N/O και μετά enter: ');
readln(symbol)
until (symbol = 'o')
or (symbol = 'O')
end.
```

## ◆ Ποιοτικά χαρακτηριστικά προγραμμάτων

- Αναγνωσιμότητα
  - απλότητα
  - κατάλληλη επιλογή ονομάτων, π.χ.  
`monthly_income`      `incomeBeforeTaxes`
  - στοίχιση
  - σχόλια
- Φιλικότητα προς το χρήστη
- Τεκμηρίωση
- Συντήρηση
- Ενημέρωση

## ◆ Στοίχιση

- Πρόγραμμα και υποπρογράμματα

**program** ...  
δηλώσεις

**begin**  
εντολές  
**end.**

**procedure** ...  
δηλώσεις

**begin**  
εντολές  
**end**

**function** ...  
δηλώσεις

**begin**  
εντολές  
**end**

- Απλές εντολές

**if** ... **then**  
εντολή

**else**  
εντολή

**while** ... **do**  
εντολή

**for** ... **do**  
εντολή

## ◆ Στοίχιση (συνέχεια)

- Σύνθετες εντολές

**if ... then  
begin**

εντολές

**end**

**else**

**begin**

εντολές

**end**

**while ... do  
begin**

εντολές

**end**

**for ... do  
begin**

εντολές

**end**

## ◆ Στοίχιση (συνέχεια)

- Σύνθετες εντολές (συνέχεια)

**repeat**

εντολές

**until** ...

**case** ... **of**

τιμή<sub>1</sub> : εντολή<sub>1</sub>;

τιμή<sub>2</sub> : εντολή<sub>2</sub>;

...

τιμή<sub>n</sub> : εντολή<sub>n</sub>

**end**

**with** ... **do**

**begin**

εντολές

**end**

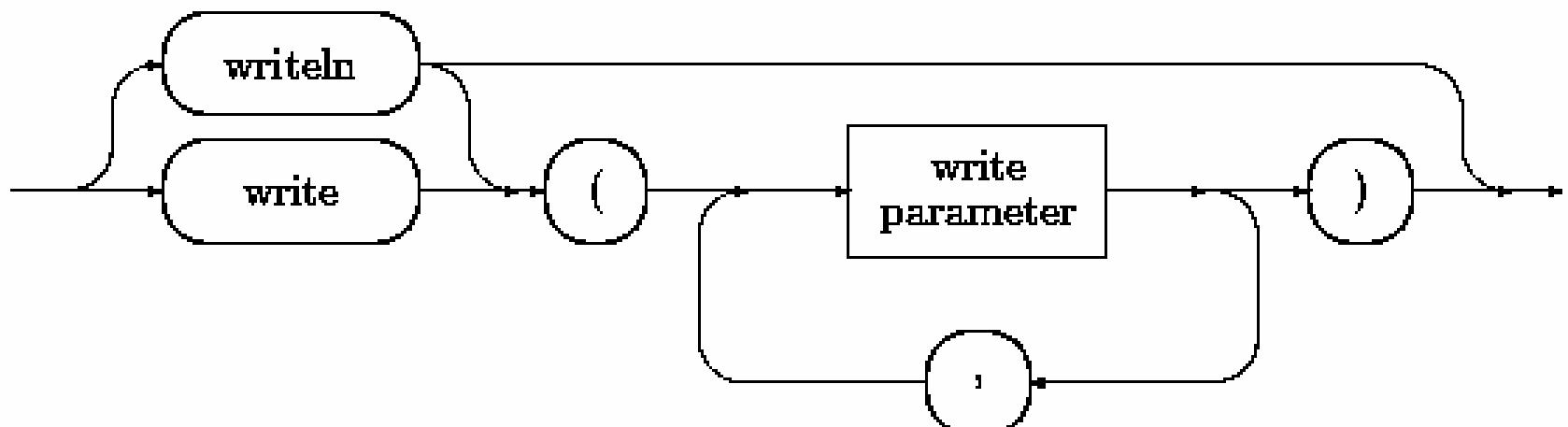
# Είσοδος και έξοδος

---

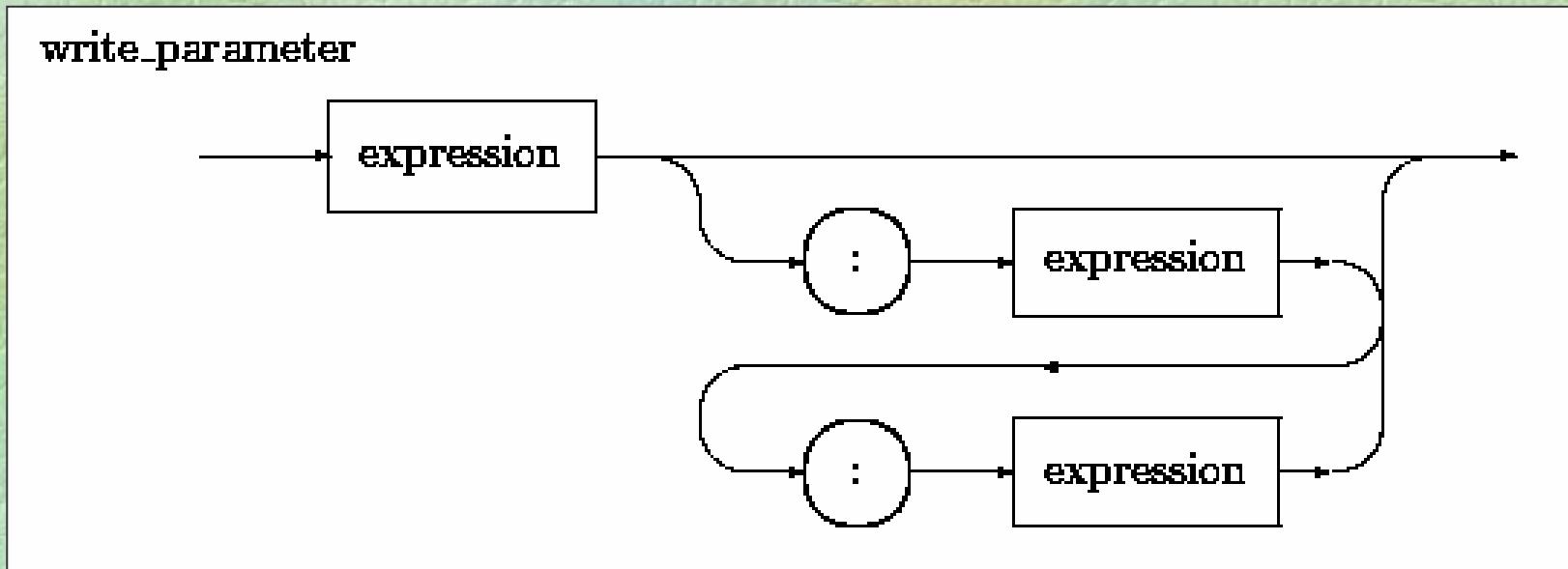
- ◆ Συσκευές εισόδου/εξόδου
- ◆ Είσοδος: εισαγωγή δεδομένων
  - π.χ. από το πληκτρολόγιο
- ◆ Έξοδος: παρουσίαση αποτελεσμάτων
  - π.χ. στην οθόνη
- ◆ Προκαθορισμένες διαδικασίες της Pascal
  - **read, readln**
  - **write, writeln**

## ◆ Έξοδος με μορφοποίηση

`write_statement`



## ◆ Έξοδος με μορφοποίηση (συνέχεια)



- ◆ Συμβολοσειρές, χωρίς μορφοποίηση

```
program stars(input, output);
  var i, j : integer;
begin
  for i := 0 to 5 do
    begin
      for j := 1 to 2*i do write('*');
      writeln('$')
    end
end.
```

\$  
\*\*\$  
\*\*\*\*\$  
\*\*\*\*\*\$  
\*\*\*\*\*\$  
\*\*\*\*\*\$

## ◆ Ακέραιες τιμές, χωρίς μορφοποίηση

```
program example(input, output);
  var x, y, sum : integer;
begin
  x := 3; y := 6;
  sum := x + y;
  writeln(x, '+', y, '=', sum)
end.
```



- ◆ Ακέραιες τιμές, χωρίς μορφοποίηση

```
program indent(input, output);  
begin  
    writeln('x =', 42);  
    writeln('y =', -324);  
    writeln('z =', 123837)  
end.
```

<b>x =</b>	<b>42</b>
<b>y =</b>	<b>-324</b>
<b>z =</b>	<b>123837</b>

- ◆ Ακέραιες τιμές, με μορφοποίηση

`writeln(42:4)`

		4	2
--	--	---	---

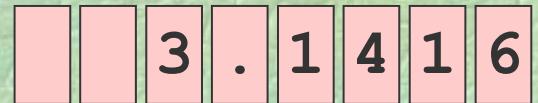
- ◆ Το προηγούμενο παράδειγμα

```
program example(input, output);  
  var x, y, sum : integer;  
begin  
  x := 3; y := 6;  
  sum := x + y;  
  writeln(x:0, '+', y:0, '=', sum:0)  
end.
```

3+6=9

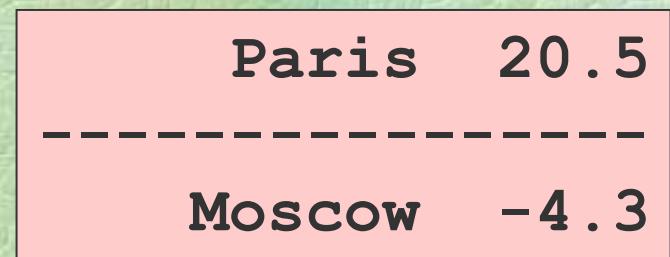
- ◆ Πραγματικές τιμές, με μορφοποίηση

```
writeln(3.14159:8:4)
```



- ◆ Παράδειγμα

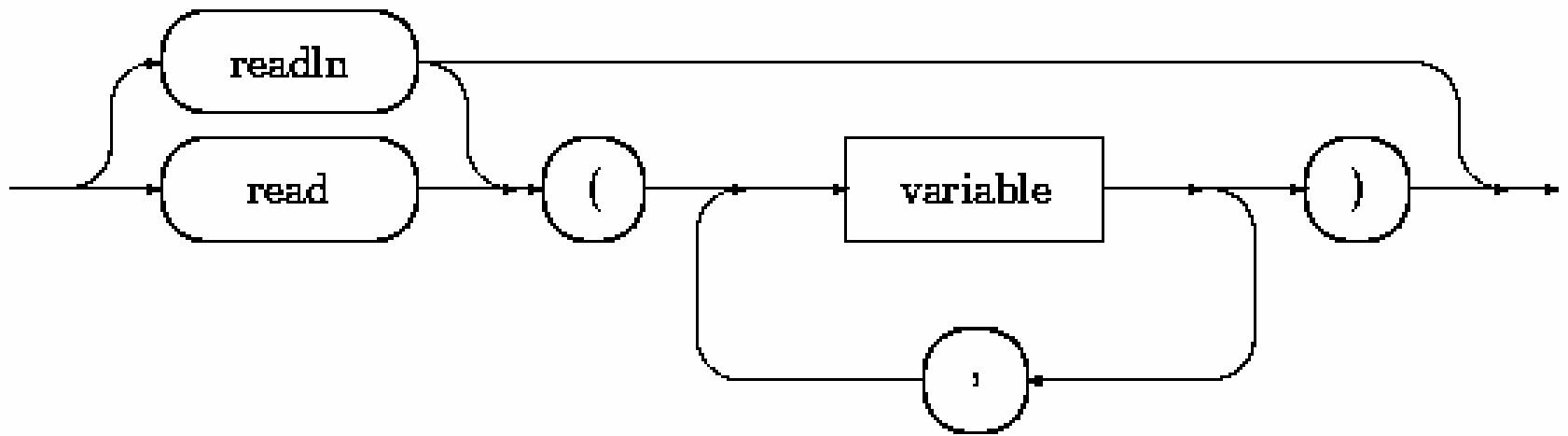
```
program temperatures(input,output);
  var i : integer; par, mos : real;
begin par := 20.5; mos := -4.3;
  writeln('Paris':10, par:6:1);
  for i:=1 to 14 do write('-'); writeln;
  writeln('Moscow':10, mos:6:1)
end.
```



Paris	20.5
-----	
Moscow	-4.3

## ◆ Συντακτικό διάγραμμα

read\_statement



## ◆ 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

## ◆ Παραδείγματα

## ◆ Παράδειγμα

```
program example5(input,output);
  var x, y, sum : integer;
begin
  writeln('Θα προσθέσω δύο ακεραίους');
  write('Δώσε το x και <enter>: ');
  read(x);
  write('Δώσε το y και <enter>: ');
  read(y);
  sum := x + y;
  writeln(x:0, '+', y:0, '=', sum:0)
end.
```

## ◆ Πρώτη εκτέλεση παραδείγματος

Θα προσθέσω δύο ακεραίους

Δώσε το x και <enter>: 3 <enter>

Δώσε το y και <enter>: 6 <enter>

$3+6=9$

## ◆ Δεύτερη εκτέλεση παραδείγματος

Θα προσθέσω δύο ακεραίους

Δώσε το x και <enter>: 3 6 <enter>

Δώσε το y και <enter>: 3+6=9

- ◆ Αρχεία εισόδου και εξόδου
- ◆ Παράδειγμα με αρχείο εισόδου

```
program test1(input, output, F);
  var f : text;
begin
  reset(f);
  ...
  read(f, ...);
  ...
  close(f)
end.
```

## ◆ Παράδειγμα με αρχείο εξόδου

```
program test2(input, output, G) ;
    var g : text;
begin
    rewrite(g);
    ...
    write(g, ...);
    ...
    close(g)
end.
```

## ◆ Είδη ορθότητας

- Συντακτική
- Νοηματική
- Σημασιολογική

## ◆ Σημασιολογική ορθότητα ελέγχεται:

- με δοκιμές (testing)
- με μαθηματική επαλήθευση

## ◆ Παράδειγμα: εύρεση γινομένου

```
function mult(x,y : integer) : integer;
    var i,z : integer;
begin
    z := 0;
    for i := 1 to x do z := z+y;
    mult := z
end
```

## ◆ Ισχυρισμός:

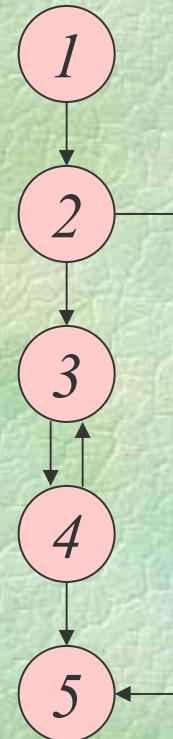
- Η συνάρτηση υπολογίζει το γινόμενο δυο φυσικών αριθμών x και y

- ◆ Εντοπισμός σημείων όπου θα γραφούν βεβαιώσεις

```

function mult(x,y : integer) : integer;
    var i,z : integer;
begin (*1*)
    z := 0; (*2*)
    for i := 1 to x do (*3*)
        z := z+y (*4*) ;
    (*5*) mult := z
end

```



- ◆ Καταγραφή όλων των δυνατών τρόπων ροής ελέγχου

## ◆ Βεβαιώσεις

(\*1 – Βεβαίωση εισόδου:  $x \geq 0, y \geq 0$  \*)

**z := 0;**

(\*2 :  $x \geq 0, y \geq 0, z = 0$  \*)

**for i:=1 to x do**

(\*3 – Αναλλοίωτη βρόχου:

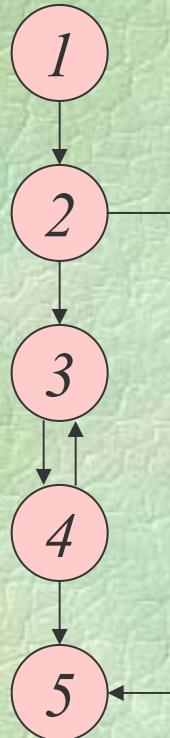
$x \geq 0, y \geq 0, i \leq x, z = y * (i-1)$  \*)

**z := z+y**

(\*4 :  $x \geq 0, y \geq 0, z = y * i$  \*) ;

(\*5 – Βεβαίωση εξόδου:  $x \geq 0, y \geq 0, z = y * x$  \*)

**mult := 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

```
function power(y:real; j:integer):real;
    var x,z:real; i:integer;
begin (*1*) x:=y; i:=j; (*2*)
    if i<0 then
begin (*3*) x:=1/x; i:=abs(i) end;
    (*4*) z:=1;
    while i>0 do
begin (*5*) if odd(i) then z:=z*x;
    (*6*) x:=sqr(x); i:=i div 2 (*7*)
end;
    (*8*) power:=z
end
```

◆ Ροή ελέγχου

◆ Βεβαιώσεις

(\*1 – Βεβαίωση εισόδου:  $x : \text{real}$ ,  $y : \text{integer} *$ )

(\*2 :  $x = y$ ,  $i = j *$ )

(\*3 :  $i < 0 *$ )

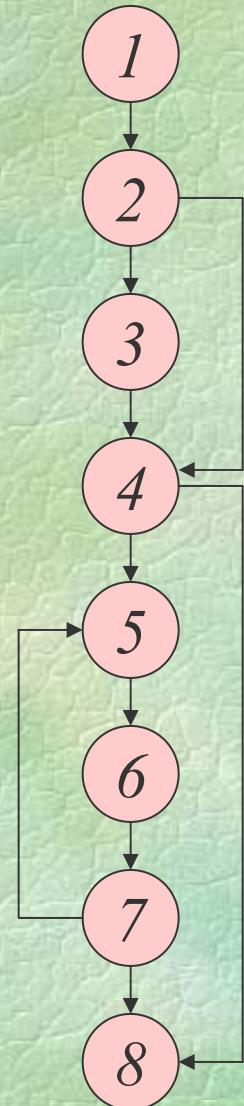
(\*4 :  $i \geq 0$ ,  $y^j = x^i *$ )

(\*5 – Αναλλοίωτη βρόχου:  $i \geq 0$ ,  $y^j = z * x^i *$ )

(\*6 :  $i \geq 0$ ,     $y^j = z * x^i$     αν  $i$  άρτιος,  
                  $y^j = z * x^{i-1}$     αν  $i$  περιττός \*)

(\*7 :  $y^j = z * x^i *$ )

(\*8 – Βεβαίωση εξόδου:  $y^j = z *$ )



## ◆ 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

# Τακτικοί τύποι

---

◆ Οι τύποι **integer**, **boolean** και **char**

◆ Απαριθμητοί τύποι

```
type color = (white, red, blue, green,  
              yellow, black, purple);  
sex     = (male, female);  
day     = (mon, tue, wed, thu, fri,  
              sat, sun);  
  
var c : color; d : day;
```

◆ Πράξεις με τακτικούς τύπους

- συναρτήσεις **succ**, **pred**, **ord**
- τελεστές σύγκρισης **=**, **<>**, **<**, **>**, **<=**, **>=**

# Τύποι υποπεριοχής

- ◆ Ορίζουν υποπεριοχές τακτικών τύπων

```
type grade    = 0..10;
      digit   = '0'..'9';
      workday = mon..fri;

var i : -10..10;
```

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

`var n : array [1..5] of integer;`

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

`n[1], n[2], n[3], n[4], n[5]`

και έχουν τύπο `integer`

## ◆ Παραδείγματα

```
var a : array [1..10] of real;  
      b : array ['a'..'z'] of integer;  
      c : array [mon..sun] of char;  
  
....  
  
a[1] := 4.2;  
readln(a[3]);  
a[10] := a[1];  
  
b['n'] := b['x']+1;  
  
c[tue] := 't'
```

## ◆ Διάβασμα ενός πίνακα

- γνωστό μέγεθος

```
for i:=1 to 10 do read(a[i])
```

- πρώτα διαβάζεται το μέγεθος

```
read(howmany) ;
```

```
for i:=1 to howmany do read(a[i])
```

- στα παραπάνω πρέπει να προηγηθούν

```
var a : array [1..10] of real;  
i, howmany : 1..10;
```

## ◆ Διάβασμα ενός πίνακα (συνέχεια)

- τερματισμός με την τιμή 0  
`read(x); i:=0;`  
`while x<>0 do`  
`begin i:=i+1; a[i]:=x; read(x) end`
- στο παραπάνω πρέπει να προηγηθούν  
`var a : array [1..10] of real;`  
`i : 0..10; x : real;`
- 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 i:=1 to 10 do writeln(a[i]);
```

```
if a[k] > a[k+1] then ...
```

## ◆ Αρχικοποίηση (με μηδενικά)

```
for i:=1 to 10 do a[i]:=0
```

```
for ch:='a' to 'z' do b[ch]:=0
```

## ◆ Εύρεση ελάχιστου στοιχείου

```
x := a[1];
```

```
for i:=2 to 10 do
```

```
    if a[i] < x then x := a[i]
```

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

```
program search(input,output);
var x : integer;
     a : array [1..10] of integer;
     άλλες δηλώσεις;
begin τίτλος επικεφαλίδα;
     οδηγίες στο χρήστη ;
     read(x);
     διάβασμα του πίνακα ;
     ψάξιμο στον πίνακα για τον x ;
     παρουσίαση αποτελεσμάτων
end.
```

## ◆ Μια δυνατή συγκεκριμενοποίηση

```
for i:=1 to 10 do read(a[i]);  
i:=0;  
repeat i:=i+1 until (a[i]=x) or (i=10);  
  
if a[i]=x then  
    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)

## ◆ Εναλλακτική συγκεκριμενοποίηση #1

```
i:=0;  
repeat i:=i+1;  
    if a[i]=x then found:=true  
                  else found:=false  
until found or (i=10);  
  
if found then  
    writeln('Το βρήκα στη θέση ', i)  
else  
    writeln('Δεν το βρήκα')
```

## ◆ Εναλλακτική συγκεκριμενοποίηση #2

```
i:=0; found:=false;  
repeat i:=i+1;  
    if a[i]=x then found:=true  
until found or (i=10);  
  
if found then  
    writeln('Το βρήκα στη θέση ', i)  
else  
    writeln('Δεν το βρήκα')
```

## ◆ Εναλλακτική συγκεκριμενοποίηση #3

```
i:=0;  
repeat i:=i+1; found := a[i]=x  
until found or (i=10);  
  
if found then  
    writeln('Το βρήκα στη θέση ', i)  
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

```
program binsearch(input,output);
  const n=100;
  var i,howmany,mid,first,last : 0..n;
      a : array [1..n] of integer;
      x : integer; found : boolean;
begin Μήνυμα επικεφαλίδα και οδηγίες χρήσης;
  read(howmany); (* κατά αύξονσα σειρά *)
  for i:=1 to howmany do read(a[i]);
  read(x);
Αναζήτηση και εμφάνιση αποτελέσματος
end.
```

## ◆ Αναζήτηση και εμφάνιση αποτελέσματος

```
first:=1; last:=howmany;  
found := false;  
while not found and (first<=last) do  
begin mid := (first+last) div 2;  
    found := x=a[mid];  
    if x<a[mid] then last:=mid-1  
        else first:=mid+1  
end;  
if found then writeln(mid)  
else writeln('not found')
```

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

```
procedure swap(var x, y : integer);  
    var save : integer;  
begin  
    save:=x; x:=y; y:=save  
end
```

## ◆ Μέθοδος της φυσαλίδας

```
for i:=1 to n-1 do
    for j:=n-1 downto i do
        if a[j] > a[j+1] then
            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

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

$i = 1$ : 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

$i = 2$ : 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

$i = 3$ : 4 5 6 12 7 9 8

---

4 5 6 12 7 8 9

4 5 6 12 7 8 9

$i = 4$ : 4 5 6 7 12 8 9

---

4 5 6 7 12 8 9

$i = 5$ : 4 5 6 7 8 12 9

---

$i = 6$ : 4 5 6 7 8 9 12

---

# Πολυδιάστατοι πίνακες

## ◆ Παράδειγμα

```
var a : array [1..10,5..16] of integer;  
...  
a[1,13] := 42;  
for i:=1 to 10 do  
  for j:=5 to 16 do read(a[i,j])
```

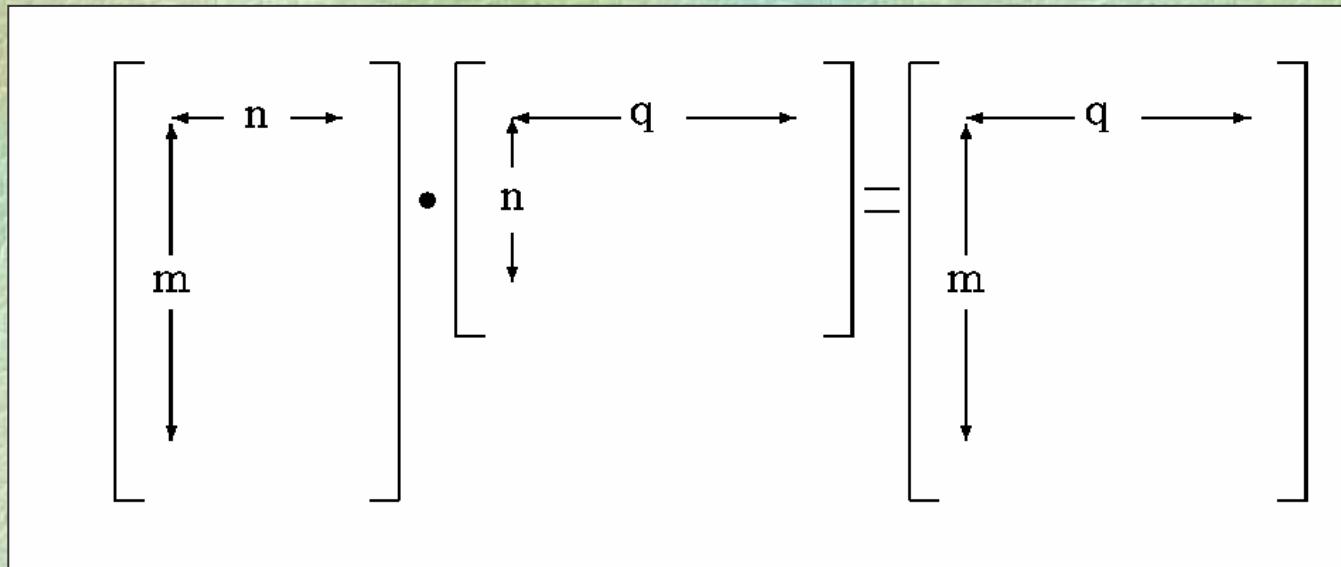
## ◆ Ισοδύναμος ορισμός και χρήση

```
var a : array [1..10] of  
              array [5..16] of integer;  
... a[i][j] ...
```

# Πολλαπλασιασμός πινάκων (i)

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

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



## ◆ Το πρόγραμμα

```
var a : array [1..m,1..n] of real;  
      b : array [1..n,1..q] of real;  
      c : array [1..m,1..q] of real;  
  
...  
  
for i:=1 to m do  
  for j:=1 to q do  
    begin  
      c[i,j] := 0;  
      for k:=1 to n do  
        c[i,j] := c[i,j] + a[i,k]*b[k,j]  
    end
```

◆ 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								

0								

3								

1								

1								

1								

1								

8	2							

8	2							

8	2							

8	2							

## ◆ Κατασκευή για περιπτώ $n$

```
i:=n div 2; j:=n; k:=0;  
for h:=1 to n do  
begin j:=j-1; a[i,j]:=k; k:=k+1;  
    for m:=2 to n do  
        begin j:=(j+1) mod n; i:=(i+1) mod n;  
            a[i,j]:=k; k:=k+1  
    end  
end;  
  
for i:=0 to n-1 do  
begin  
    for j:=0 to n-1 do write(a[i,j]:4);  
    writeln  
end
```

### ◆ Τύπος **real**

- προσεγγίσεις πραγματικών αριθμών
- **trunc**: ακέραιο μέρος (αποκοπή)
- **round**: στρογγυλοποίηση

### ◆ Παράσταση κινητής υποδιαστολής

- mantissa και εκθέτης  $\pm m \cdot 2^x$   
όπου  $0.5 \leq m < 1$  και  $x \in \mathbb{Z}$  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 Newton

- 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 (ii)

## ◆ Ακολουθία προσεγγίσεων

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

## ◆ 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 (iii)

```
function sqroot(x : real) : real;
  const eps = 0.00001;      (* 1E-5 *)
  var old, new : real;
begin
  new := 1;
  repeat
    old := new;
    new := (old + x/old) / 2
  until (* συνθήκη τερματισμού *);
  sqroot := new
end
```

# Εύρεση τετραγωνικής ρίζας (iv)

## ◆ Εναλλακτικές συνθήκες τερματισμού

- Σταθερός αριθμός επαναλήψεων  
 $n = 20$
- Επιτυχής εύρεση ρίζας                                  λάθος!  
 $sqr(new) = x$
- Απόλυτη σύγκλιση  
 $abs(sqr(new) - x) < eps$
- Σχετική σύγκλιση  
 $abs(sqr(new) - x) / new < eps$

# Εύρεση τετραγωνικής ρίζας (v)

## ◆ Εναλλακτικές συνθήκες τερματισμού

- Απόλυτη σύγκλιση κατά Cauchy  
 $\text{abs}(\text{new-old}) < \text{eps}$
- Σχετική σύγκλιση  
 $\text{abs}(\text{new-old}) / \text{new} < \text{eps}$

# Προκαθορισμένες συναρτήσεις

Συναρτήσεις	Τύπος ορίσματος	Τύπος αποτελέσματος
abs, sqr <b>(απόλυτο) (τετράγωνο)</b>	integer, real	ίδιος
sin, cos, exp, ln <b>(ημ) (συν) (εκθ) (log)</b>	integer, real	real
sqrt, arctan <b>(ρίζα) (τοξεφ)</b>		
odd <b>(περιττός)</b>	integer	boolean
eof, eoln	text	boolean
trunc, round	real	integer
succ <b>(επόμενος)</b> pred <b>(προηγούμενος)</b>	integer, boolean, char	ίδιος
ord <b>(χωδικός ASCII)</b> <b>(αντίστροφη της ord)</b>	char integer	integer char

# Γραφικές παραστάσεις (i)

## ◆ Μορφοποίηση εξόδου (επανάληψη)

- `write(i:15)`
- `write(' | ':40)`
- `write(x:10:4)`

## ◆ Γραφικές παραστάσεις με χαρακτήρες

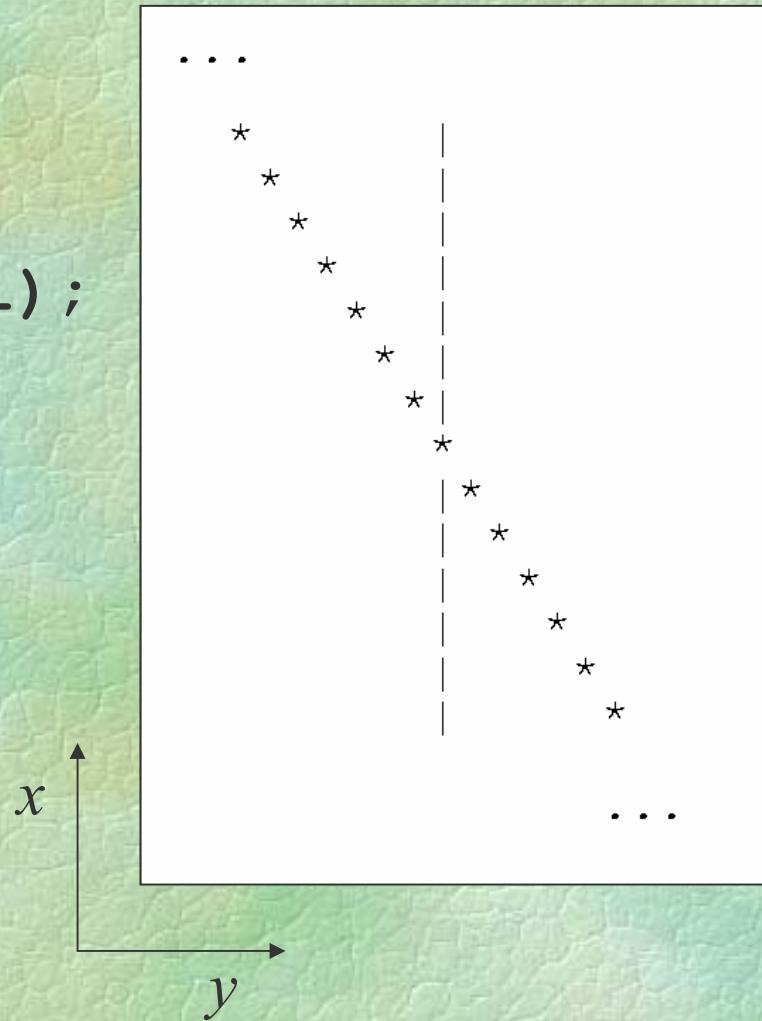
- Συνάρτηση  $y = f(x)$
- Συνήθως μας βολεύει να έχουμε τον άξονα των  $x$  κατακόρυφο και τον άξονα των  $y$  οριζόντιο

# Γραφικές παραστάσεις

(ii)

◆ Παράδειγμα:  $f(x) = -x$

```
for i:=1 to 39 do
    writeln('*':i, '|':40-i);
writeln('*':40);
for i:=1 to 39 do
    writeln(' '|40, '*'':i)
```



◆ Παράδειγμα:  $f(x) = 18 \sin x + 15 \cos 2x$

```
program graph(output);
var k,n : integer; pi : real;
procedure axis;
  var i : integer;
begin
  for i := 1 to 79 do write('-');
  writeln
end;
```

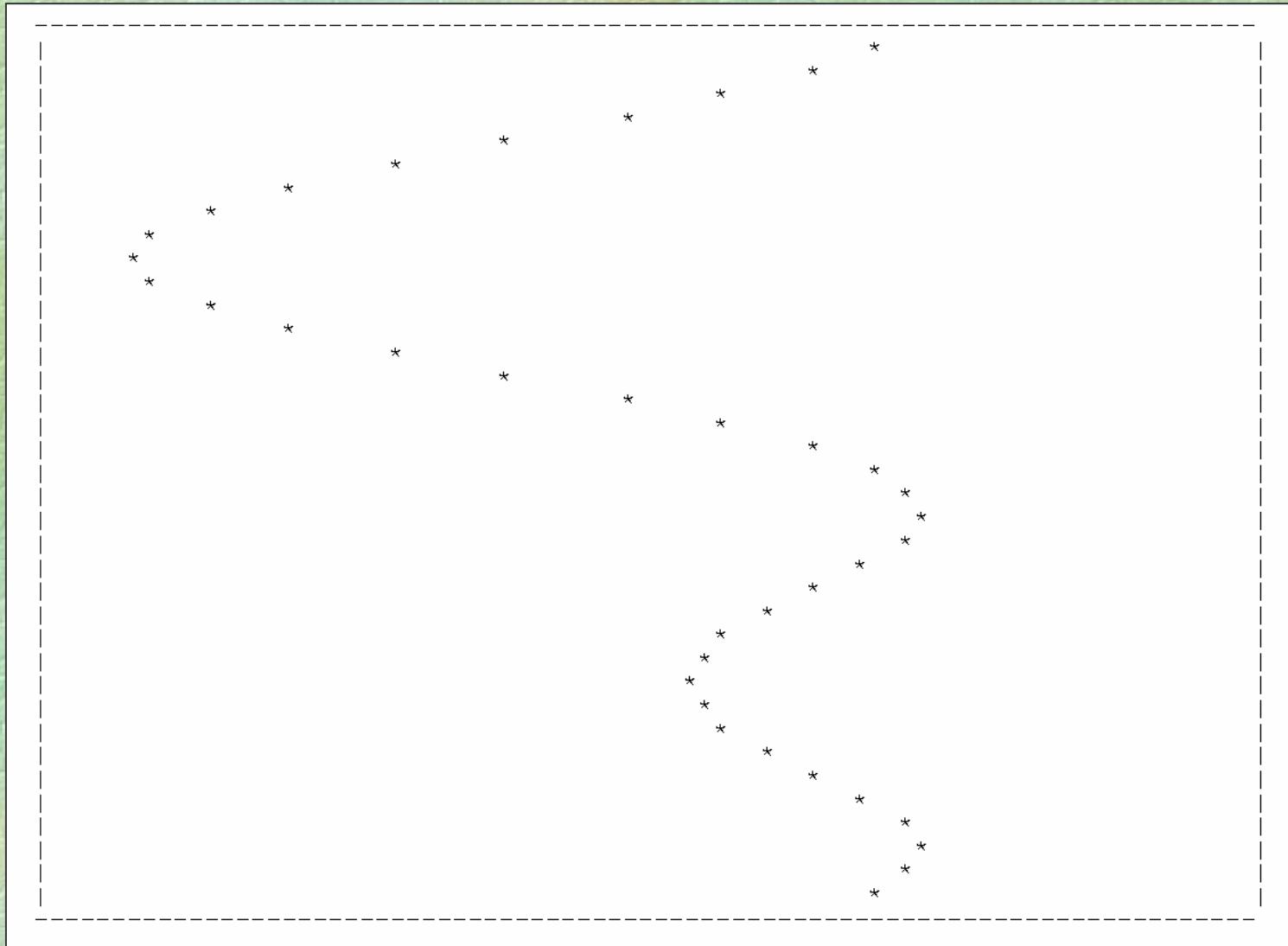
## ◆ Παράδειγμα (συνέχεια)

```
function f(j : integer) : integer;
    var x,y : real;
begin x := pi * j / 18;
    y := 18 * sin(x) + 15 * cos(2*x);
    f := round(y)
end;

begin pi := 4 * arctan(1); axis;
    for n := -18 to 18 do
begin k := f(n) + 40;
    writeln(' | ', '*' :k, ' | ':79-k)
end;
axis
end.
```

# Γραφικές παραστάσεις

(v)



◆ Παράδειγμα:  $f(x) = (3 \cos x) / x$

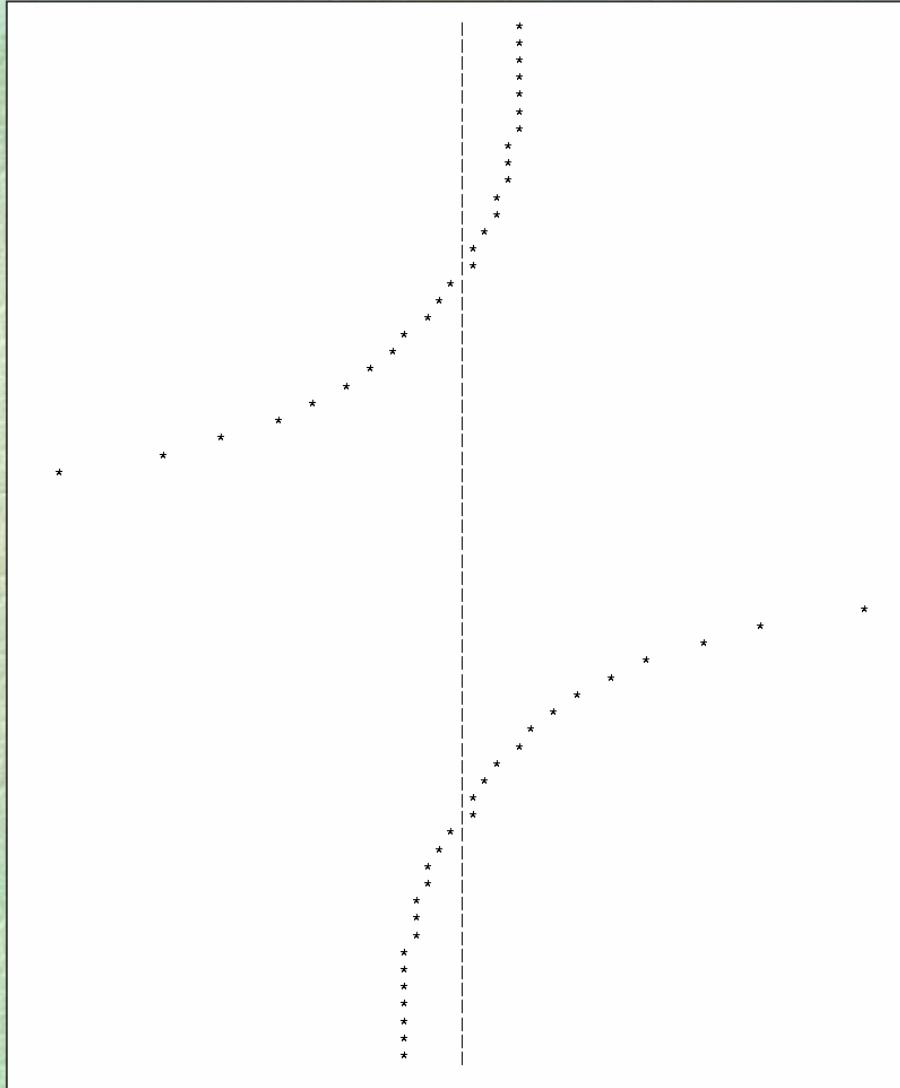
```
program printplot(output);
  const deltax = 0.1;
  bound = 30; wid = 39;
  scale = 5; shift0 = 40;
  var i : -bound..bound;
      n : integer; x : real;
  function f(x : real) : real;
    const eps = 1E-5;
    huge = (* μεγάλος αριθμός *);
begin
  if abs(x)<eps then f := huge
  else f := 3 * cos(x) / x
end;
```

## ◆ Παράδειγμα (συνέχεια)

```
begin
    x := - bound * deltax;
    for i := - bound to bound do
        begin
            n := round(scale*f(x));
            if abs(n)>wid then
                writeln ('|':shift0)
            else if n<0 then
                writeln('*':n+shift0,'|':-n)
            else
                writeln('|':shift0,'*':n);
            x := x + deltax
        end
    end.
```

# Γραφικές παραστάσεις

(viii)



◆ Παράδειγμα:  $f(x)$  και  $g(x)$

```
program twocurves(output);
const ...;
var ...;
    line: array[-wid..wid] of char;
function one(...); ...
function two(...); ...

begin
    for j:=-wid to wid do line[j]:=' ';
    line[0]:='|';
    ...
end.
```

## ◆ Παράδειγμα (συνέχεια)

```
for i := ...
begin
    n := ...one...; m := ...two...
    line[n] :='*'; line[m] :='.';
    for j:=-wid to wid do
        write(line[j]);
    writeln;
    line[n] := ' '; line[m] := ' ';
    line[0] := '|';
    x := x + delta
end
end.
```

# 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)

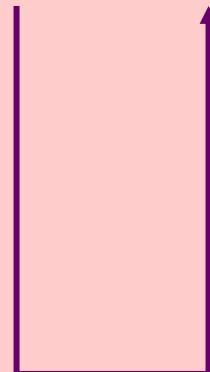
```
function mycos(x : real) : real;  
  const eps = 1E-5;  
  var sqx, term, sum : real;  
      n : integer;  
  
begin n := -1; sqx := sqr(x);  
  term := 1; sum := 1;  
  repeat n := n + 2;  
    term := -term * sqx / (n*(n+1));  
    sum := sum + term  
  until abs(term/sum) < eps;  
  mycos := sum  
end
```

- ◆ 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)

```
function fact(n : integer) : integer;  
begin  
    if n=0 then fact := 1  
            else fact := fact(n-1) * n  
end
```

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

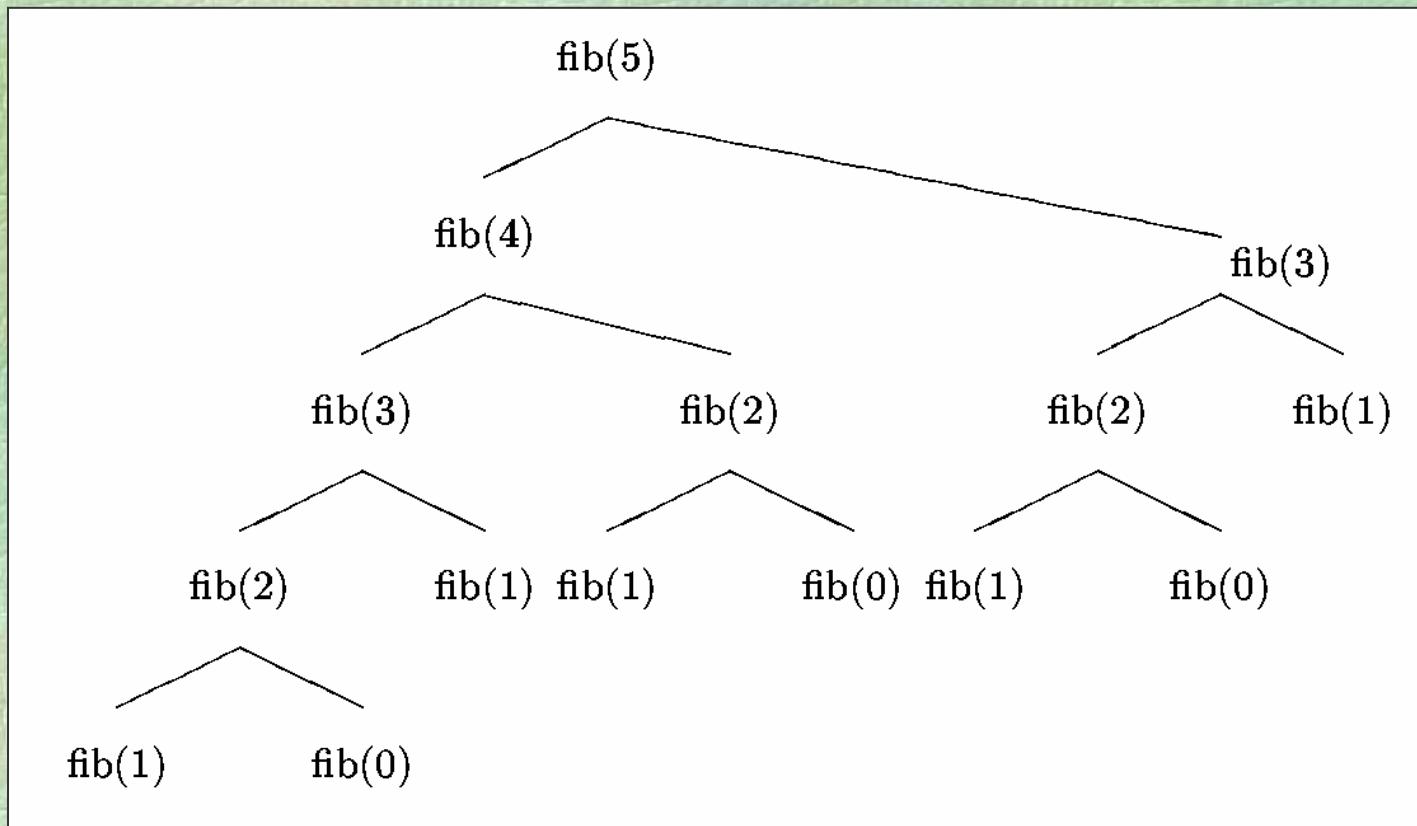
## ◆ Αριθμοί Fibonacci

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

## ◆ Αναδρομική συνάρτηση υπολογισμού

```
function fib(n : integer) : integer;  
begin  
    if (n=0) or (n=1) then  
        fib := 1  
    else  
        fib := fib(n-1) + fib(n-2)  
    end
```

- ◆ Αυτός ο αναδρομικός υπολογισμός των αριθμών Fibonacci δεν είναι αποδοτικός



## ◆ Μέγιστος κοινός διαιρέτης

- Αναδρομική υλοποίηση του αλγορίθμου του Ευκλείδη

```
function gcd(i,j : integer) : integer;
begin
    if (i=0) or (j=0) then
        gcd := i+j
    else if i > j then
        gcd := gcd(i mod j, j)
    else
        gcd := gcd(i, j mod i)
end
```

## ◆ Συνάρτηση παρόμοια με του Ackermann

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

$$z(i, 0, 1) = i$$

$$z(i, 0, 2) = 0$$

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

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

```

function z (i,j,n : integer) : integer;
begin
    if n=0 then z:=j+1
    else if j=0 then
        if n=1 then z:=i
        else if n=2 then z:=0
                    else z:=1
    else z:=z (i,z (i,j-1,n) ,n-1)
end
```

# Αμοιβαία αναδρομή

```
function f2(n:integer):integer; forward;  
  
function f1(n:integer):integer;  
begin  
    if n=0 then f1 := 5  
            else f1 := f1(n-1) * f2(n-1)  
end  
  
function f2(n:integer):integer;  
begin  
    if n=0 then f2 := 3  
            else f2 := f1(n-1) + 2*f2(n-1)  
end
```

- ◆ Μέθοδος φυσαλλίδας (BubbleSort)  
με έλεγχο εναλλαγών

```
i:=0;  
repeat i := i + 1; noswaps := true;  
    for j := n-1 downto i do  
        if a[j] > a[j+1] then  
            begin swap(a[j],a[j+1]);  
                noswaps := false  
            end  
until noswaps
```

- ◆ Στην καλύτερη περίπτωση απαιτούνται  $O(n)$  συγκρίσεις, στη χειρότερη  $O(n^2)$

## ◆ Ταξινόμηση με συγχώνευση (MergeSort)

- 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

```
procedure mergesort(var a : list;
                     fa, la : integer);
  var b : list; i, mid : integer;
begin
  if fa<la then
    begin mid := (fa + la) div 2;
      mergesort(a, fa, mid);
      mergesort(a, mid+1, la);
      merge(a, a, b, fa, mid, mid+1,
            la, fa, la);
      for i := fa to la do a[i]:=b[i]
    end
  end
```

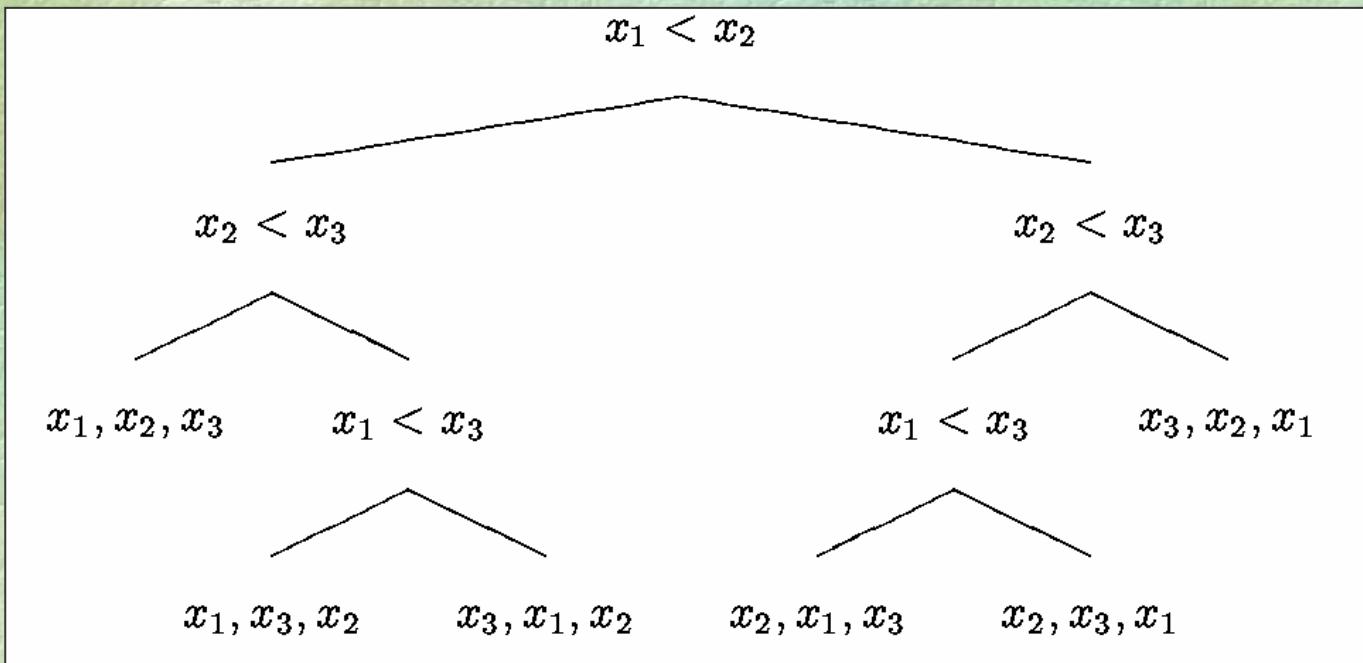
## ◆ Συγχώνευση

```
procedure merge(var a,b,c : list;
                fa,la,fb,lb,fc : integer;
                var lc : integer);
    var ia, ib, ic : integer;
begin
    ia := fa; ib := fb; ic := fc;
repeat
    if a[ia]<b[ib] then
        begin c[ic]:=a[ia]; ia:=ia+1 end
    else
        begin c[ic]:=b[ib]; ib:=ib+1 end;
    ic := ic+1
until (ia>la) or (ib>lb);
```

## ◆ Συγχώνευση (συνέχεια)

```
for ia := ia to la do
    begin c[ic]:=a[ia]; ic:=ic+1 end;
for ib := ib to lb do
    begin c[ic]:=b[ib]; ic:=ic+1 end;
lc := ic-1
end
```

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



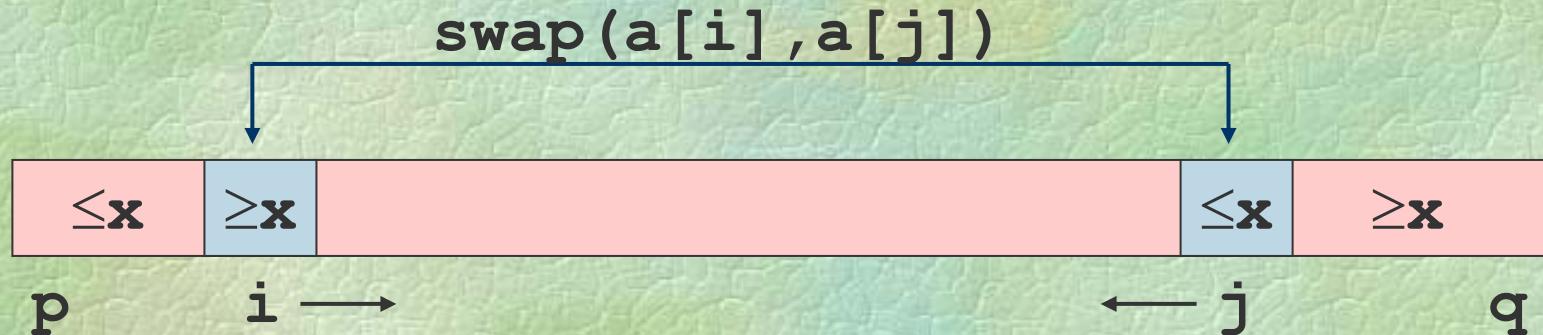
## ◆ Ταξινόμηση με διαμέριση (Quicksort)

```
if p < q then begin
    partition(p, q, i, j);
    quicksort(p, j); quicksort(i, q)
end
```

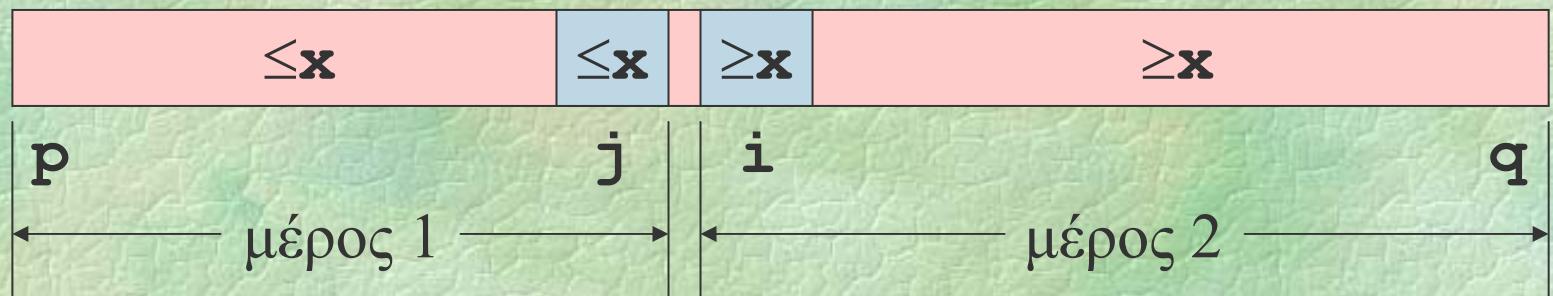
## ◆ Διαμέριση (partition)

```
επιλογή ενός x := a[k];
repeat
    while a[i] < x do i := i+1;
    while x < a[j] do j := j-1;
    if i <= j then begin
        swap(a[i],a[j]); i := i+1; j := j-1
    end
until i > j
```

- ◆ Σε κάθε βήμα της διαμέρισης



- ◆ Μετά τη διαμέριση

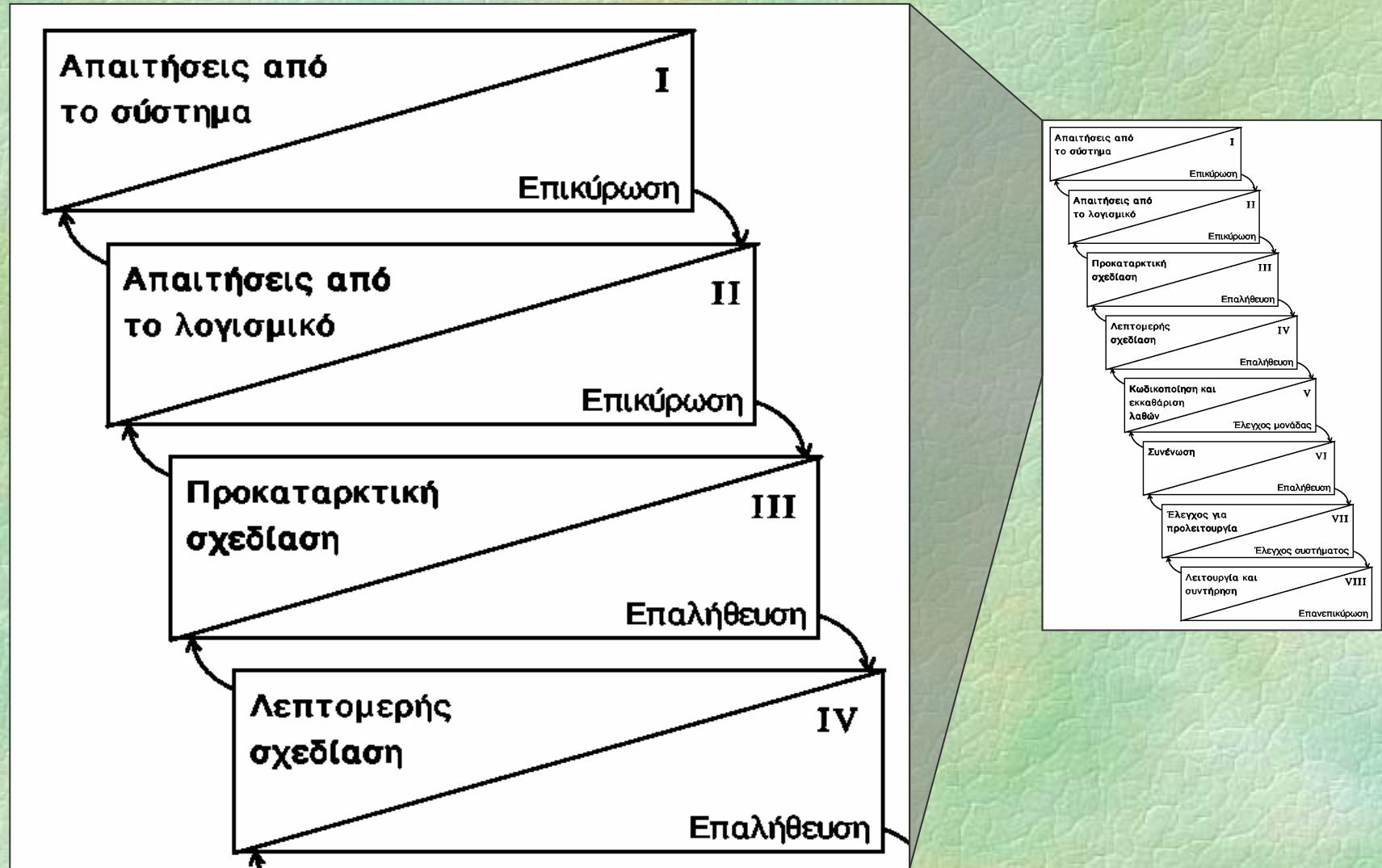


# Τεχνολογία λογισμικού

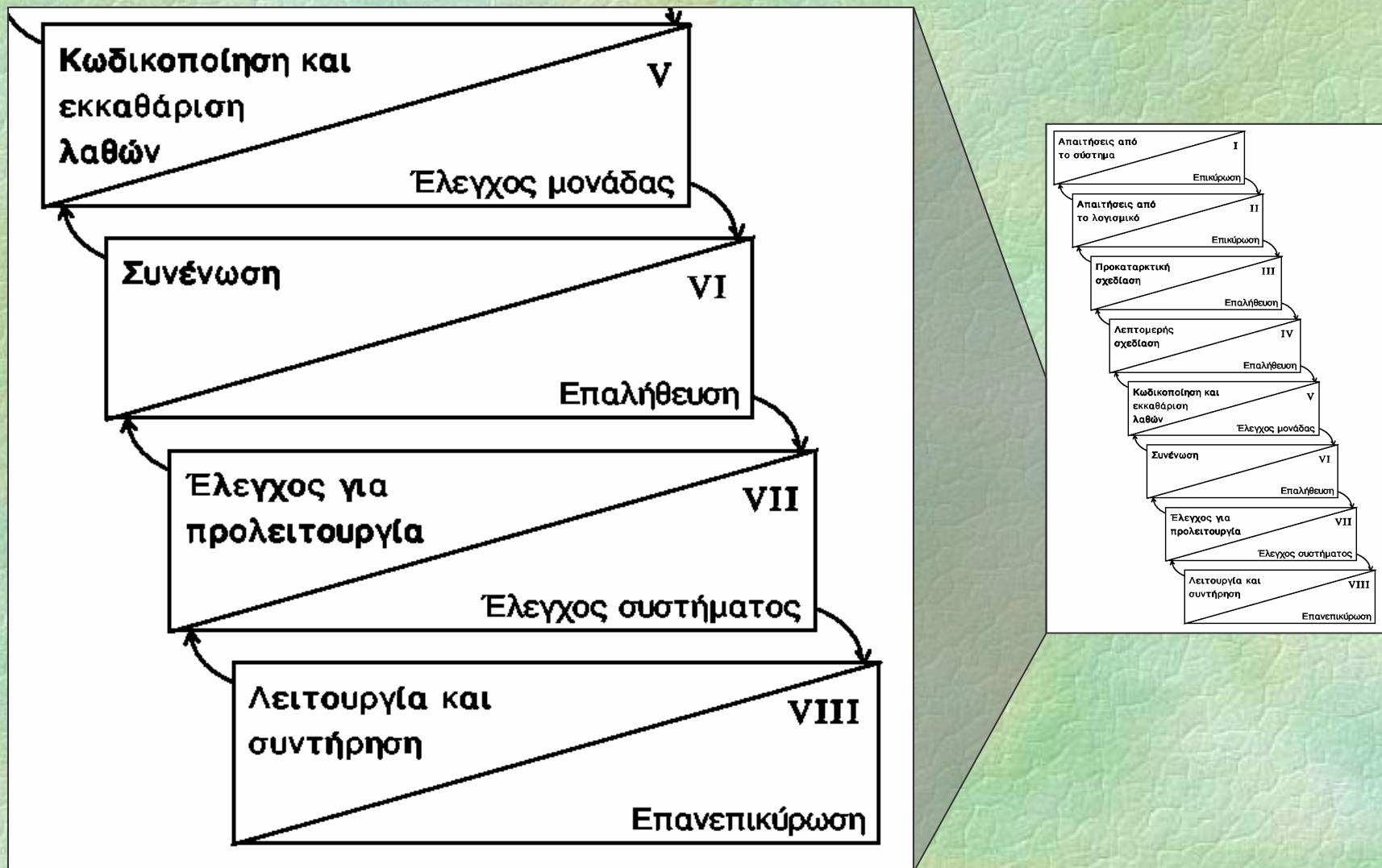
- ◆ Software engineering
- ◆ Ανάπτυξη λογισμικού που να εξασφαλίζει:
  - παράδοση μέσα σε προδιαγεγραμμένα χρονικά όρια
  - κόστος μέσα σε προδιαγεγραμμένα όρια
  - καλή ποιότητα
  - αξιοπιστία
  - δυνατή και όχι δαπανηρή συντήρηση
- ◆ Μοντέλα κύκλου ζωής λογισμικού

# Μοντέλο του καταρράκτη

(i)



# Μοντέλο του καταρράκτη (ii)



# Πέρασμα παραμέτρων, εμβέλεια (i)

- ◆ Φώλιασμα υποπρογραμμάτων:  
υποπρογράμματα περιέχουν άλλα  
υποπρογράμματα
- ◆ Σύγκρουση ονομάτων: το ίδιο όνομα  
δηλώνεται σε πολλά υποπρογράμματα
- ◆ Κανόνες εμβέλειας: εξηγούν κάθε όνομα  
που εμφανίζεται στο πρόγραμμα σε ποια  
δήλωση αντιστοιχεί
- ◆ Γενικά και τοπικά ονόματα
- ◆ Παράμετροι τιμών και μεταβλητών

## ◆ Κανόνες εμβέλειας της Pascal

- ενότητα: πρόγραμμα ή υποπρόγραμμα
- οι ενότητες είναι δυνατόν να **περιέχουν** άλλες ενότητες
- κάθε όνομα **ορίζεται** σε κάποια ενότητα
- **εμβέλεια** ενός ονόματος είναι η ενότητα μέσα στην οποία ορίζεται, αλλά:
- η εμβέλεια ενός ονόματος δεν περιέχει τυχόν ενότητες όπου το όνομα αυτό **επανορίζεται**

# Πέρασμα παραμέτρων, εμβέλεια (iii)

## ◆ Παράδειγμα

```
program p(input,output);
var a,b,c,d:integer;

procedure p1(a:integer; var b:integer);
var c:integer;
begin c:=b; d:=2*a; b:=c+a; a:=c+b;
writeln(a:5,b:5,c:5,d:5)
end
```

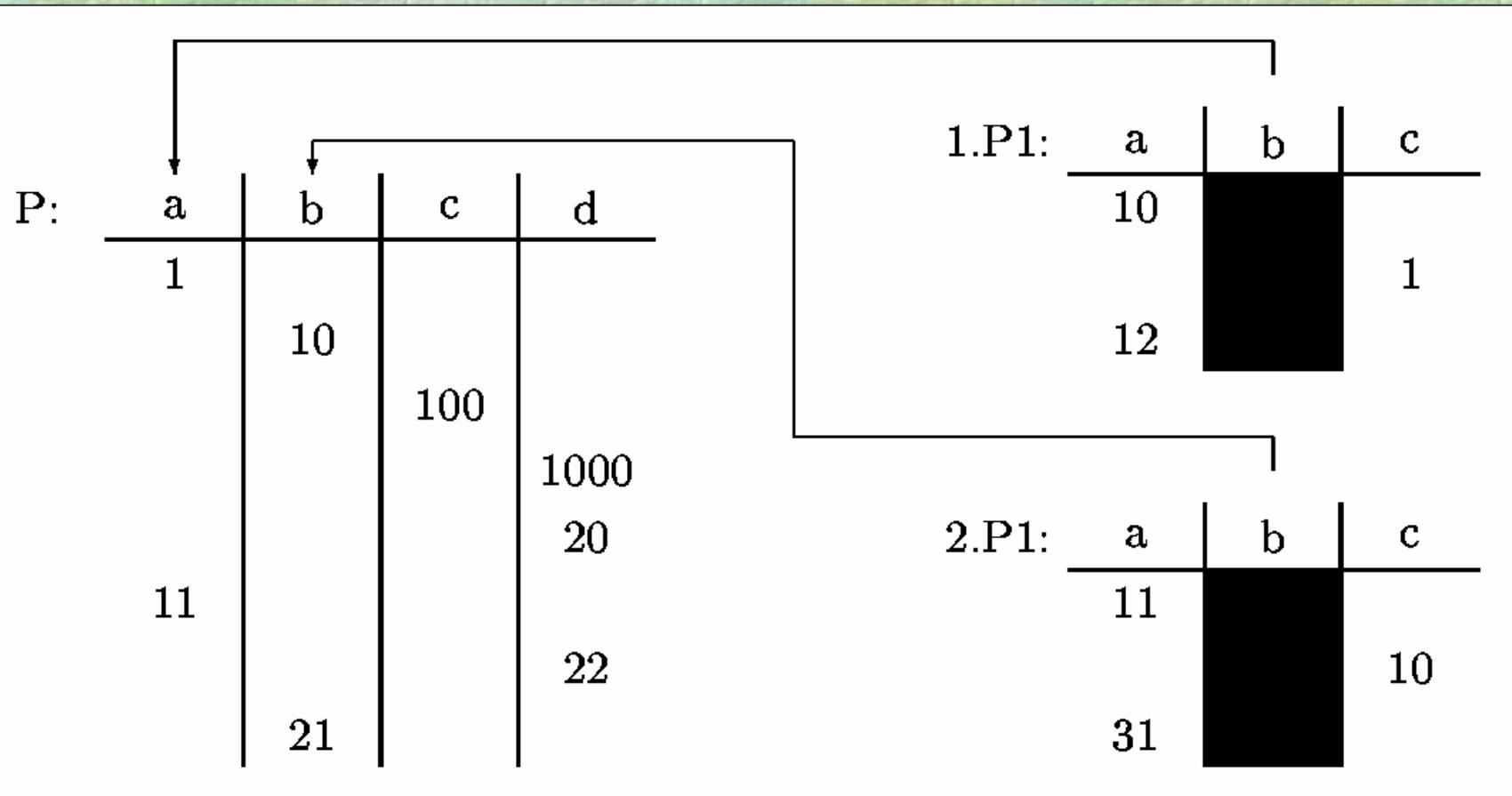
Ενότητα 2

```
begin a:=1; b:=10; c:=100; d:=1000;
writeln(a:5,b:5,c:5,d:5);
p1(b,a); writeln(a:5,b:5,c:5,d:5);
p1(a,b); writeln(a:5,b:5,c:5,d:5)
end.
```

Ενότητα 1

# Πέρασμα παραμέτρων, εμβέλεια (iv)

## ◆ Παράδειγμα, εκτέλεση με το χέρι



# Πέρασμα παραμέτρων, εμβέλεια (v)

- ◆ Παράδειγμα, εκτέλεση με το χέρι (αποτελέσματα)

	1	10	100	1000
	12	11	1	20
<u>Output:</u>	11	10	100	20
	31	21	10	22
	11	21	100	22

## ◆ Σημειώσεις – παρατηρήσεις

- επανάληψη της λέξης **var**  
**procedure p(var r, s : integer;**  
                 **var done : boolean);**
- στη λίστα των παραμέτρων επιτρέπονται μόνο αναγνωριστικά τύπων  
**procedure p(var a : array [1..30]**  
                 **of integer);**      λάθος!  
  
**type list = array [1..30] of integer;**  
**procedure p(var a : list);**                 σωστό
- να αποφεύγονται παράμετροι τιμών **array**

# Επεξεργασία κειμένου (i)

- ◆ Διάβασμα και επεξεργασία όλων των χαρακτήρων που περιέχονται σε ένα αρχείο

```
while not eof(fil) do  
begin read(fil,ch); process(ch) end
```

- ◆ Διάβασμα και επεξεργασία όλων των ακεραίων που περιέχονται σε ένα αρχείο

```
read(fil,i);  
while not eof(fil) do  
begin process(i); read(fil,i) end
```

- ◆ Η συνάρτηση **eof** επιστρέφει **true** αν φτάσαμε στο τέλος του αρχείου

## ◆ Παράδειγμα 1: πρόγραμμα που

- διαβάζει ένα κείμενο από το αρχείο FIL
- μετράει τον αριθμό των χαρακτήρων και τον αριθμό των γραμμών
- υπολογίζει το μέσο όρο μήκους γραμμής

```
program tp(input, output, FIL) ;
δηλώσεις ;
begin
    τίτλος και οδηγίες ;
    επεξεργασία κειμένου ;
    παρουσίαση των αποτελεσμάτων
end.
```

### ◆ Παράδειγμα 1: επεξεργασία κειμένου

*αρχικοποίηση ;*

**while** **not** **eof(fil)** **do**

**begin**

*επεξεργασία μιας γραμμής ;*

**linecount := linecount + 1**

**end**

### ◆ Παράδειγμα 1: επεξεργασία μιας γραμμής

**while** **not** **eoln(fil)** **do**

**begin**

**read(fil,ch) ;**

**charcount := charcount + 1**

**end;**

**readln(fil)**

## ◆ Παράδειγμα 1: αρχικοποίηση

```
reset(fil);  
linecount := 0; charcount := 0;
```

## ◆ Παράδειγμα 1: παρουσίαση αποτελεσμάτων

```
writeln('charcount =', charcount);  
writeln('linecount =', linecount);  
if linecount > 0 then  
  writeln('mean length =',  
         charcount div linecount)
```

## ◆ Παράδειγμα 2: πρόγραμμα που

- διαβάζει ένα κείμενο από το πληκτρολόγιο
- μετράει τον αριθμό των χαρακτήρων, τον αριθμό των λέξεων και τον αριθμό των γραμμών

## ◆ Συνάρτηση για τον εντοπισμό γραμμάτων

```
function letter(ch : char) : boolean;  
begin  
    letter := (ch >= 'a') and (ch <= 'z')  
        or (ch >= 'A') and (ch <= 'Z')  
end
```

## ◆ Παράδειγμα 2

```
lets:=0; words:=0; lines:=0;  
while not eof do begin  
    while not eoln do begin  
        read(ch);  
        if letter(ch) then begin  
            while not eoln and letter(ch) do  
                begin lets:=lets+1; read(ch) end;  
            if letter(ch) then lets:=lets+1;  
            words:=words+1  
        end  
    end;  
    lines:=lines+1; readln  
end
```

## ◆ Παράδειγμα 3: πρόγραμμα που

- διαβάζει ένα κείμενο από το αρχείο FIL
- μετράει τις συχνότητες εμφάνισης λέξεων με μήκος από 1 μέχρι 20
- εμφανίζει τα αποτελέσματα ως εξής:

<b>words of length</b>	1	6	*****
<b>words of length</b>	2	3	***
<b>words of length</b>	3	0	
...			
<b>words of length</b>	20	2	**

## ◆ Παράδειγμα 3

```
program wordslength(input,output,FIL);
  const max = 20;
  var fil : text;
      freq : array[1..max] of integer;
      { λοιπες δηλώσεις μεταβλητών }

function letter(ch : char) : boolean;
begin
  letter := (ch>='a') and (ch<='z')
            or (ch>='A') and (ch<='Z')
end;
```

## ◆ Παράδειγμα 3 (συνέχεια)

**begin**

```
{ τίτλος, αρχικοποίηση } ;  
while not eof(fil) do  
  begin i:=0; read(fil,ch);  
    while letter(ch) do  
      begin i:=i+1; read(fil,ch) end;  
      if (i>0) and (i<=max) then  
        freq[i] := freq[i] + 1  
end;
```

# Επεξεργασία κειμένου (x)

## ◆ Παράδειγμα 3 (συνέχεια)

```
for i:=1 to max do
begin
  write('words of length', i:3,
        freq[i]:4, ' ');
  for j:=1 to freq[i] do write('*');
  writeln
end
end.
```

## ◆ Μετατροπή κεφαλαίων σε μικρά

```
function lowercase(ch : char) : char;
begin
  if (ch>='A') and (ch<='Z') then
    lowercase := chr(ord(ch) - ord('A')
                     + ord('a'))
  else lowercase := ch
end
```

## ◆ Μετατροπή μικρών σε κεφαλαία, ομοίως

```
if (ch>='a') and (ch<='z') then
  uppercase := chr(ord(ch) - ord('a')
                     + ord('A'))
else uppercase := ch
```

## ◆ Εύρεση εμφάνισης λέξης-κλειδιού

...

(\* η λέξη-κλειδί έχει 3 χαρακτήρες \*)

**for** j:=1 **to** 3 **do** **read**(key[j]);

...

(\* έστω i το μήκος της γραμμής \*)

**for** k:=1 **to** i-2 **do**  
    **if** (line[k] = key[1]) **and**  
        (line[k+1] = key[2]) **and**  
        (line[k+2] = key[3])  
    **then** **writeln**('keyword found')

- ◆ Ο τύπος **string** ορίζει ακολουθίες χαρακτήρων (συμβολοσειρές)
- ◆ Προσοχή: δεν ορίζεται στη Standard Pascal
- ◆ Παράδειγμα

```
var name      : string[30];  
     address   : string[80];
```

...

```
readln(name); readln(address);  
writeln('My name is ', name);  
writeln('and my address is', address)
```

- ◆ Σύγκριση με λεξικογραφική διάταξη  
'ding' < 'donut'
- ◆ Προκαθορισμένες συναρτήσεις και διαδικασίες για συμβολοσειρές

**strlen**

μήκος συμβολοσειράς

**strpos**

αναζήτηση σε συμβολοσειρά

**+**

συνένωση συμβολοσειρών

**str**

τμήμα συμβολοσειράς

**strdelete**

διαγραφή τμήματος συμβολοσειράς

**strinsert**

εισαγωγή μέσα σε συμβολοσειρά

## ◆ Μήκος συμβολοσειράς

```
s := 'abcdef'; n := strlen(s)
```

6

## ◆ Αναζήτηση σε συμβολοσειρά

```
s1 := 'It is raining'; s2 := 'rain';
n := strpos(s2, s1)
```

7

## ◆ Συνένωση συμβολοσειρών

```
s1 := 'abc'; s2 := 'def';
s := s1 + s2
```

'abcdef'

## ◆ Τμήμα συμβολοσειράς

```
s1 := 'abcdef'; s2 := str(s1, 3, 2)
```

'cd'

## ◆ Διαγραφή τμήματος συμβολοσειράς

```
s1 := 'abcdef';
```

```
s2 := strdelete(s1, 3, 2)
```

'abef'

## ◆ Εισαγωγή μέσα σε συμβολοσειρά

```
s1 := 'abcdef'; s2 := '123';
```

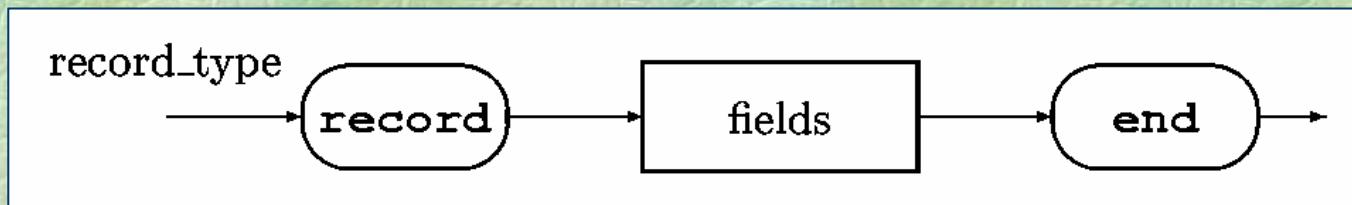
```
s := strinsert(s2, s1, 3)
```

'ab123cdef'

## ◆ Παράδειγμα

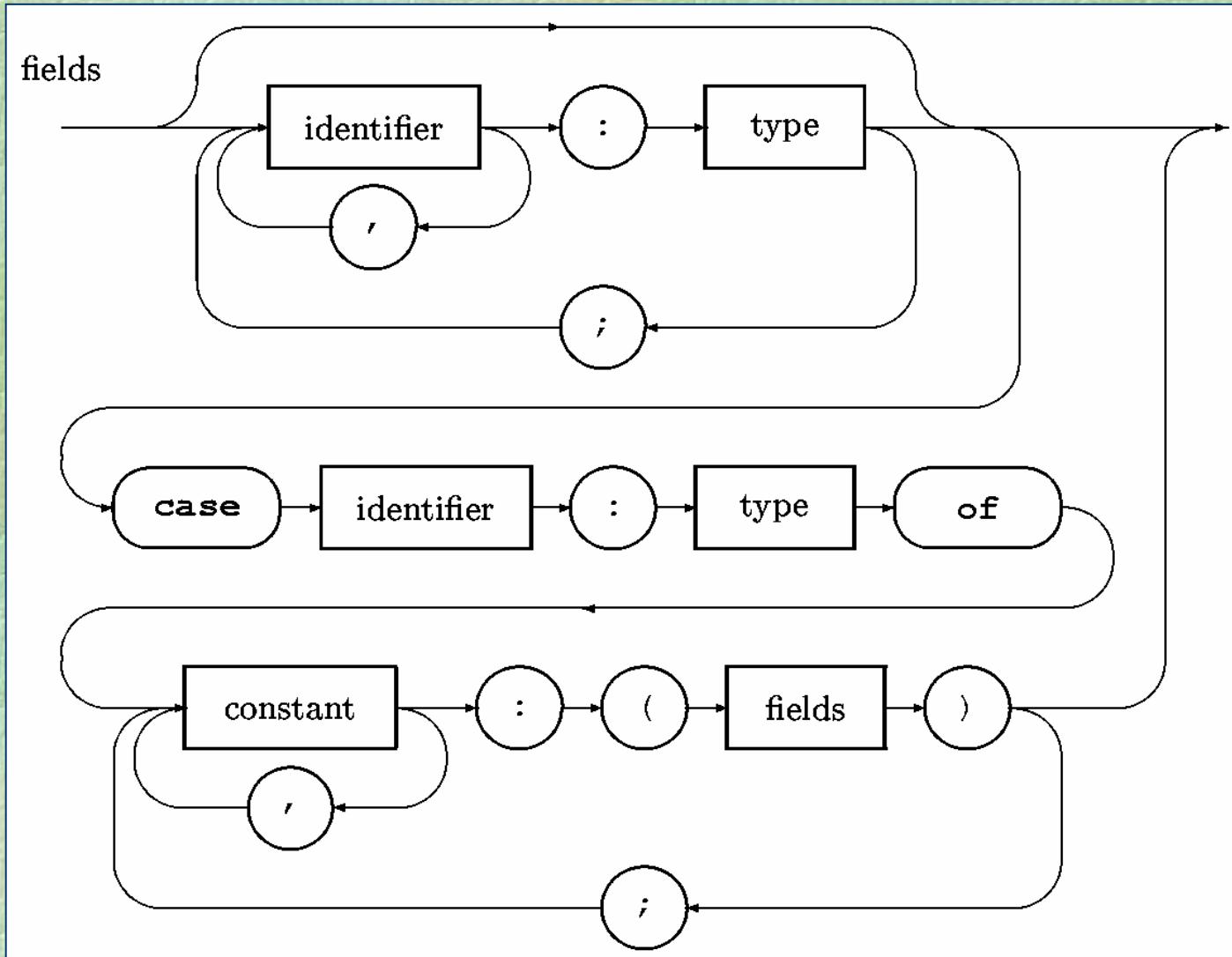
```
program AB(input, output);
  const A = 'Type in a string: ';
  var   N : integer;
        A1, A2 : string[80];
begin
  write(A); readln(A1);
  A2 := '';
  for N := strlen(A1) downto 1 do
    A2 := A2 + str(A1, N, 1);
  writeln('the reverse of ', A1);
  writeln(' is: ', A2);
  if A1 = A2 then
    writeln('palindrome')
end.
```

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



# Εγγραφές

(ii)



## ◆ Παράδειγμα

```
type StudentRecord = record
    firstName : array [1..20] of char;
    lastName  : array [1..30] of char;
    class      : 1..6;
    room       : 1..3;
    grade      : array [1..15] of 0..20
end;

var student : StudentRecord
    ...
student.class := 3;
writeln(student.firstName[1])
```

## ◆ Παράδειγμα

```
function avg(s : StudentRecord) : real;
    var sum, i : integer;
begin
    sum := 0;
    for i := 1 to 15 do
        sum := sum + s.grade[i];
    avg := sum / 15.0
end
```

# Εγγραφές μέσα σε εγγραφές (v)

```
type DateRecord = record
    day    : 1..31;
    month  : 1..12;
    year   : 1970..2100
end;

type StudentRecord = record
    ...
    birthDate : DateRecord;
    ...
end;

...
writeln(student.birthDate.day:0, '/',
       student.birthDate.month:0, '/',
       student.birthDate.year:0)
```

# Μιγαδικοί αριθμοί

```
type complex = record
    re, im : real
end;

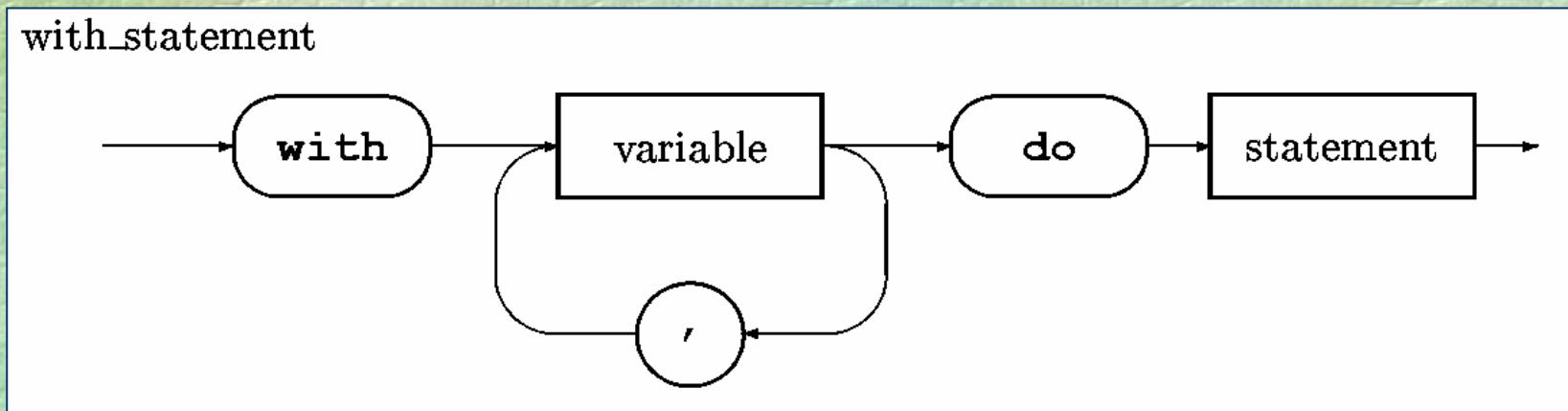
function cMult(x,y:complex) : complex;
begin
    cMult.re := x.re * y.re - x.im * y.im;
    cMult.im := x.re * y.im + x.im * y.re
end

function cNorm(c : complex) : real;
begin
    cNorm := sqrt(c.re * c.re + c.im * c.im)
end
```

# Εντολή with

(i)

- ◆ Οικονομία στην προσπέλαση των πεδίων εγγραφών
- ◆ Σύνταξη



## ◆ Παράδειγμα

```
function avg(s : StudentRecord) : real;
    var sum, i : integer;
begin
    sum := 0;
    for i := 1 to 15 do
        with s do
            sum := sum + grade[i];
    avg := sum / 15.0
end
```

# Εγγραφές με παραλλαγές

- ◆ Το πεδίο επισήμανσης καθορίζει ποια πεδία θα υπάρχουν σε μια εγγραφή
- ◆ Παράδειγμα

```
type MaritalStatus =  
    (single, married, divorced);  
  
type EmployeeRecord = record  
    name : array [1..50] of char;  
    case status : MaritalStatus of  
        single : ();  
        married, divorced :  
            (children : integer)  
end
```

- ◆ Τα στοιχεία τους πρέπει να ανήκουν σε ένα σχετικά μικρό τακτικό τύπο
- ◆ Παράδειγμα

```
type characters = set of char;
languages = set of
            (english, french, german,
             russian, greek, turkish);
numbers    = set of 1..100;
var x, y, z : languages;
...
x := [] ; y := [english] ;
z := [french, english, german] ;
```

## ◆ Πράξεις με σύνολα

- έλεγχος μέλους συνόλου                      **in**
- ένωση    **+**
- τομή    **\***
- διαφορά    **-**
- ισότητα και ανισότητα                      **= <>**
- σχέσεις υποσυνόλων                              **< > <= >=**

◆ **Αρχείο (file):** δομημένη μεταβλητή που αποτελείται από

- μεταβλητό πλήθος στοιχείων
- του ίδιου τύπου
- αποθηκευμένα το ένα μετά το άλλο
- συνήθως στην περιφερειακή μνήμη  
(π.χ. στο δίσκο)

◆ **Παράδειγμα**

```
program students(input, output, f);  
var f : file of StudentRecord
```

## ◆ Αποθηκευτική μεταβλητή (buffer)

**f^** το τρέχον στοιχείο του αρχείου

- σε περίπτωση **αρχείου εισόδου**  
το στοιχείο που μόλις διαβάστηκε
- σε περίπτωση **αρχείου εξόδου**  
το στοιχείο που πρόκειται να γραφεί

## ◆ Άνοιγμα και κλείσιμο αρχείων

**reset(f)**    **rewrite(f)**    **close(f)**

◆ Λειτουργίες εισόδου και εξόδου

**get(f)**      **put(f)**

◆ Διάβασμα και γράψιμο

**read(f,x)**     $\equiv$  **begin** **x** := **f**<sup>^</sup>; **get(f)** **end**

**write(f,x)**    $\equiv$  **begin** **f**<sup>^</sup> := **x**; **put(f)** **end**

◆ Έλεγχος τέλους αρχείου

**eof(f)**

## ◆ Παράδειγμα

```
program fileSqrt(input, output, f, g);
  var f, g : file of real;
      x      : real;
begin
  reset(f); rewrite(g);
  while not eof(f) do
  begin
    read(f, x);
    write(g, sqrt(x))
  end;
  close(f); close(g)
end.
```

## ◆ Παράδειγμα

```
program phoneDir(input, output, f);  
  
type StudentRecord = record  
    firstName : array [1..20] of char;  
    lastName : array [1..30] of char;  
    birthDate : record  
        day      : 1..31;  
        month   : 1..12;  
        year    : 1970..2100  
    end  
end  
  
var f : file of StudentRecord;
```

## ◆ Παράδειγμα (συνέχεια)

```
begin
    reset(f);
    while not eof(f) do
        begin
            with f^.birthDate do
                write(day : 0, '/',
                      month : 0, '/',
                      year : 0);
            get(f);
        end;
    close(f);
end.
```

- ◆ Τύπος **text** ισοδυναμεί με **file of char**
- ◆ Στο τέλος κάθε γραμμής υπάρχει ένας ειδικός χαρακτήρας τέλους γραμμής
  - αν διαβαστεί, αντικαθίσταται από κενό ' '
- ◆ Έλεγχοι τέλους γραμμής και τέλους αρχείου  
**eoln(f)**      **eof(f)**
- ◆ Για είσοδο και έξοδο, μπορούν να χρησιμοποιηθούν οι παρακάτω διαδικασίες  
**write(f)**            **read(f)**  
**writeln(f)**          **readln(f)**

- ◆ Παράδειγμα: μέτρηση αριθμού γραμμών και χαρακτήρων

```
program lineCount(input, output, f);
  var f                      : text;
      lines, characters : integer;
      ch                 : char;
begin lines := 0; characters := 0;
  reset(f);

  while not eof(f) do
    if eoln(f) then
      begin readln(f);
                  lines := lines + 1
      end
end
```

## ◆ Παράδειγμα (συνέχεια)

```
else
begin read(f, ch);
    characters := characters + 1
end;

close(f);

writeln('lines: ', lines);
writeln('characters: ', characters)
end.
```

- ◆ Παράδειγμα: αντιγραφή των περιεχομένων ενός αρχείου δύο φορές διαδοχικά

```
program doubleTxt(input, output, f, g);  
  var f, g : text;  
  procedure copyOnce;  
    var ch : char;  
  begin reset(f);  
    while not eof(f) do  
    begin  
      while not eoln(f) do  
      begin read(f, ch);  
        write(g, ch)  
      end;  
    end;
```

## ◆ Παράδειγμα (συνέχεια)

```
    readln(f) ;
    writeln(g)
end;
close(f)
end;

begin
    rewrite(g) ;
    copyOnce;
    copyOnce;
    close(g)
end.
```

## ◆ 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)

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

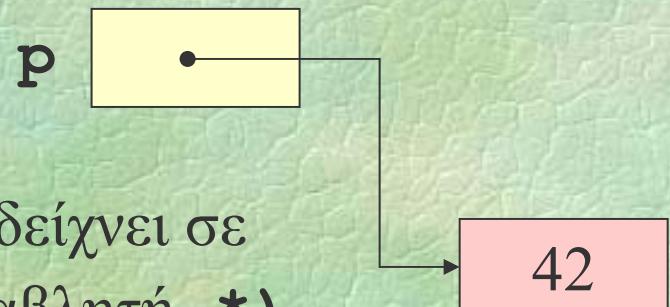
```
var p : ^integer;
```

```
...
```

(\* ο δείκτης **p** τοποθετείται να δείχνει σε κάποια ακέραια δυναμική μεταβλητή \*)

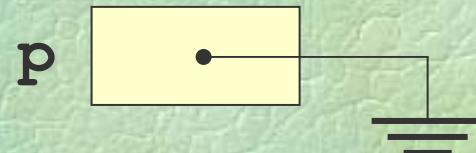
```
...
```

```
p^ := 42;  
writeln(p^ + 1)
```



- ◆ Κενός δείκτης (nil pointer): ειδική τιμή δείκτη που δε δείχνει πουθενά
- ◆ Παράδειγμα

```
var p : ^integer;  
...  
p := nil
```



- ◆ Απαγορεύεται η προσπέλαση της μνήμης μέσω ενός κενού δείκτη

```
p := nil;  
writeln(p^)
```

λάθος!

# Δυναμική παραχώρηση μνήμης

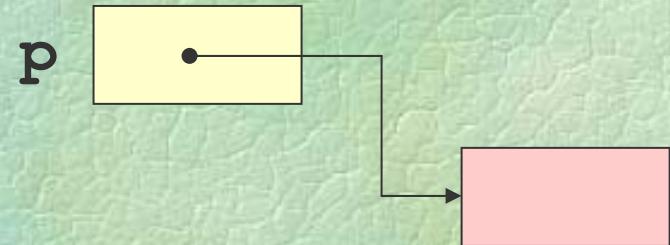
## ◆ Δέσμευση

- δημιουργία μιας νέας δυναμικής μεταβλητής

```
var p : ^integer;
```

...

```
new (p)
```

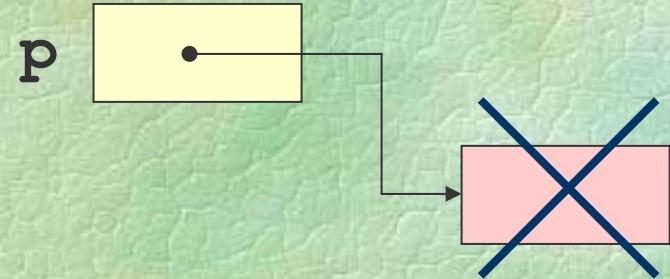


## ◆ Αποδέσμευση

- καταστροφή μιας δυναμικής μεταβλητής

...

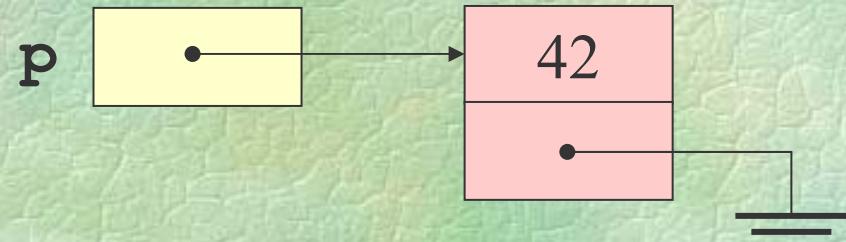
```
dispose (p)
```



# Σύνθετες δυναμικές μεταβλητές (i)

## ◆ Παράδειγμα

```
type nodeptr = ^nodetype;
nodetype = record
    info : integer;
    next : nodeptr
end;
var p : nodeptr;
...
new(p);
p^.info := 42;
p^.next := nil
```



# Σύνθετες δυναμικές μεταβλητές (ii)

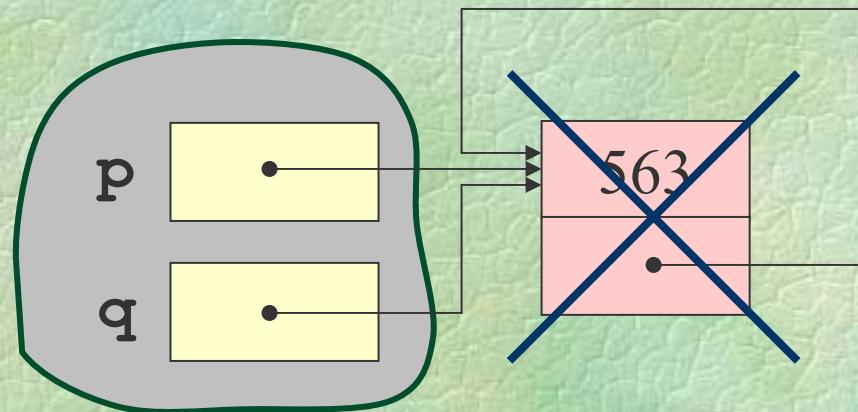
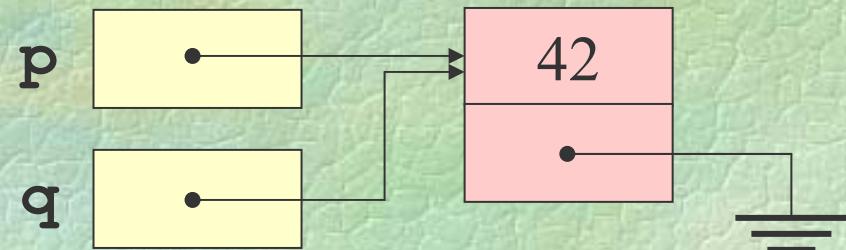
## ◆ Παράδειγμα (συνέχεια)

`q := p;`

`q^.info := 563;`

`q^.next := q;`

`dispose(p)`



ξεκρέμαστοι δείκτες!

# Δείκτες και ανάθεση

(i)

## ◆ Ανάθεση δεικτών

$q := p$

$p$



$q$



$p$

$q$



42

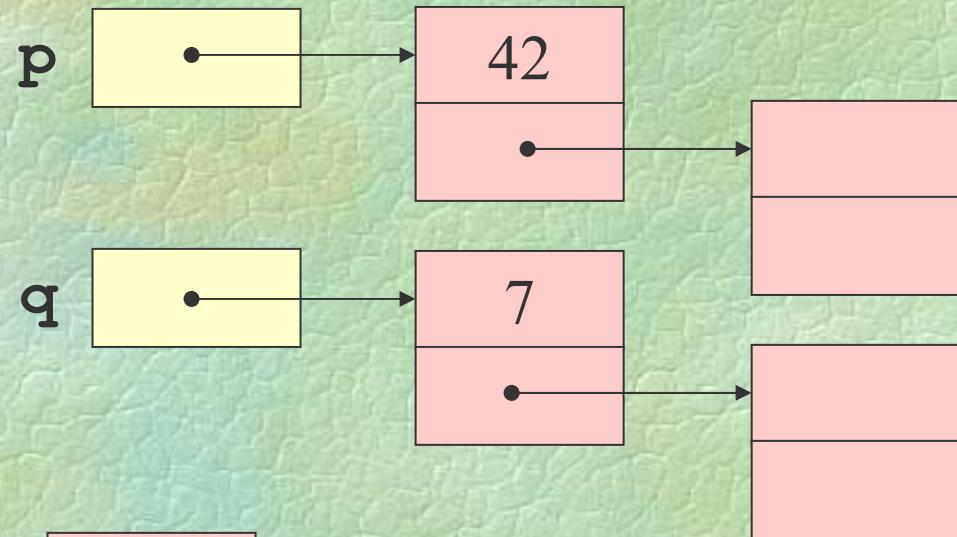
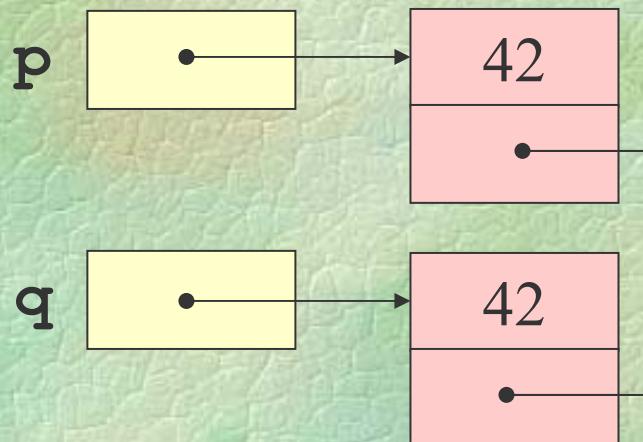
7

πριν  
μετά

σκουπίδια!

## ◆ Ανάθεση τιμών

$$q^{\wedge} := p^{\wedge}$$



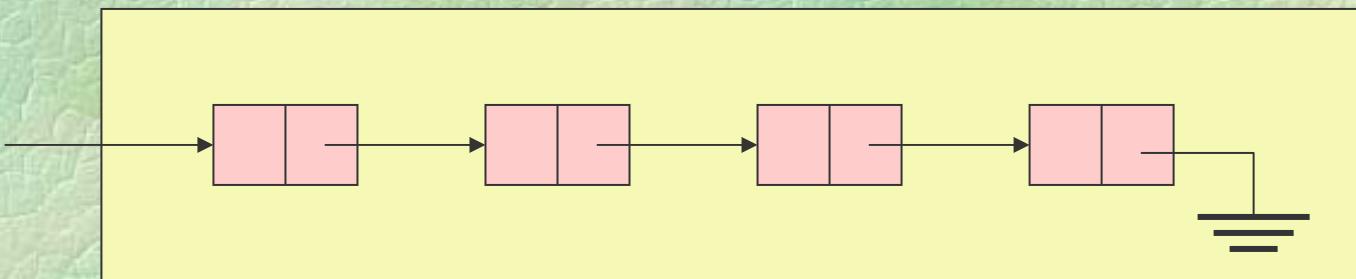
*πριν*  
*μετά*

σκουπίδια!

# Συνδεδεμένες λίστες

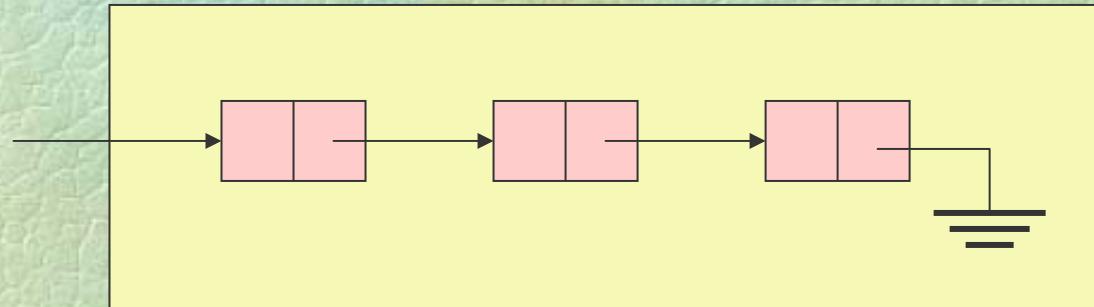
(i)

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

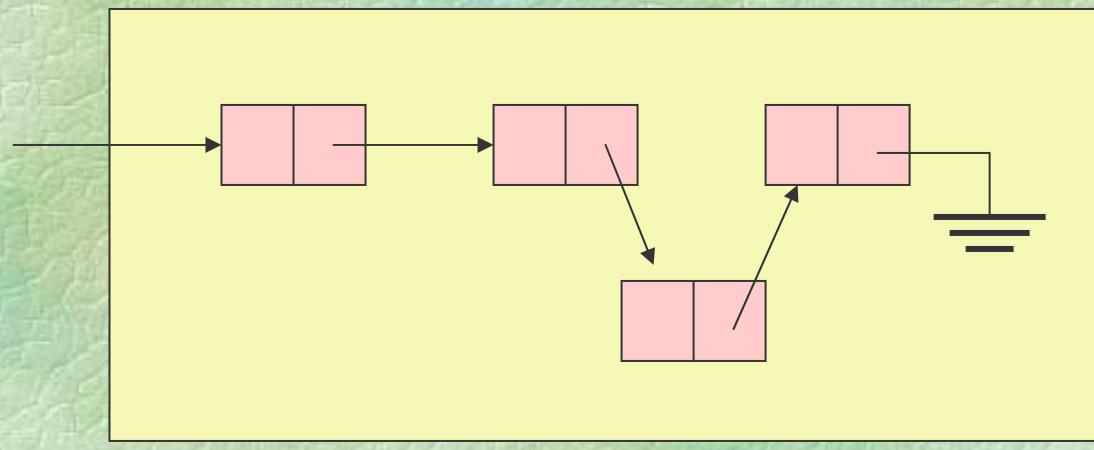


## ◆ Ευκολότερη προσθήκη στοιχείων

- πριν

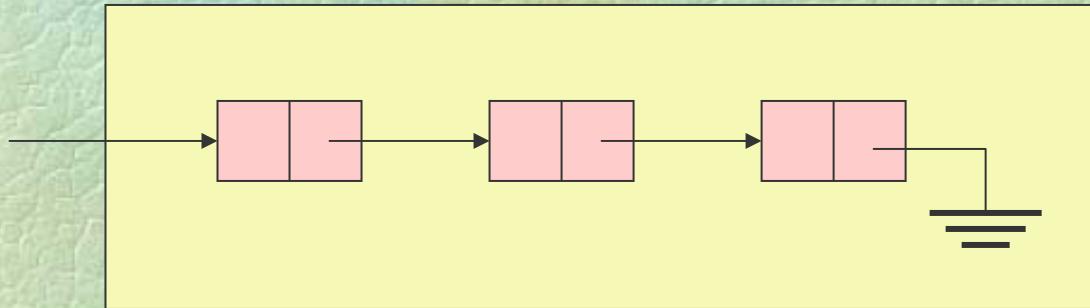


- μετά

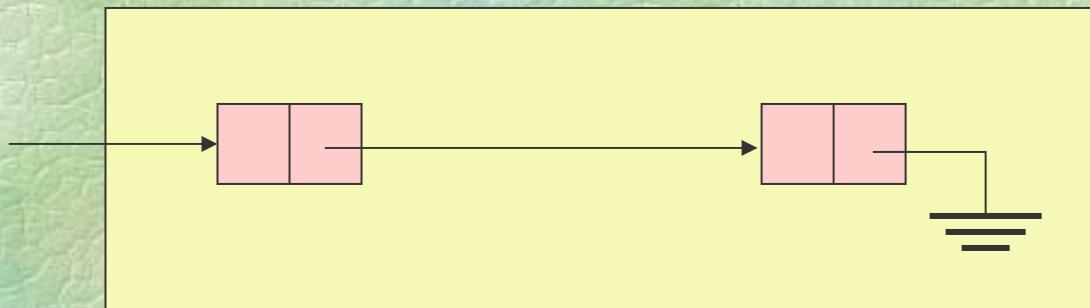


## ◆ Ευκολότερη διαγραφή στοιχείων

- πριν



- μετά



## ◆ Τύπος κόμβου συνδεδεμένης λίστας

```
type nodeptr = ^nodetype; ← πρωθύστερο!
nodetype = record
    info : integer;
    next : nodeptr
end
```

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

```
var head : nodeptr
```

## ◆ Παράδειγμα κατασκευής λίστας

```
program linkedlist(input,output);  
type nodetype = record  
    info : integer;  
    next : ^nodetype  
end;  
  
var head, node : ^nodetype;  
data : integer;
```

## ◆ Παράδειγμα (συνέχεια)

```
begin
    head := nil;
    read(data);
    while not eof do
        begin
            new(node);
            node^.info := data;
            node^.next := head;
            head := node;
            read(data)
        end
    end.
```

## ◆ Εκτύπωση λίστας

```
procedure print(p : nodeptr) ;
begin
    while p <> nil do
        begin
            writeln(p^.info) ;
            p := p^.next
        end
    end
```

## ◆ Εκτύπωση λίστας με αναδρομή

```
procedure print(p : nodeptr) ;
begin
    if p <> nil then
        begin
            writeln(p^.info) ;
            print(p^.next)
        end
    end
```

- ◆ Εκτύπωση λίστας αντίστροφα με αναδρομή

```
procedure printBackwards (p : nodeptr) ;  
begin  
    if p <> nil then  
        begin  
            printBackwards (p^.next) ;  
            writeln (p^.info)  
        end  
    end
```

◆ 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.  $f(n) = n(n-1) / 2$

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

# Πολυπλοκότητα

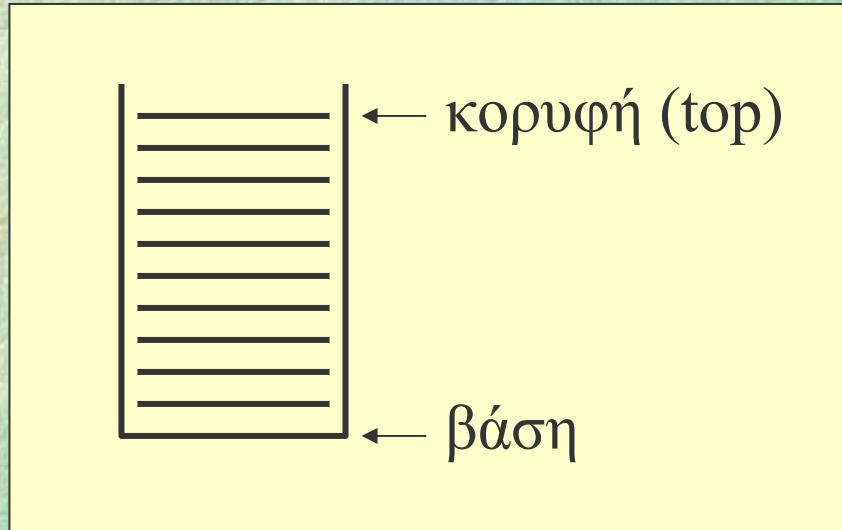
(iii)

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

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

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

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



## ◆ Αφηρημένος τύπος δεδομένων

- Ορίζεται ο τύπος **stack** που υλοποιεί τη στοίβα (ακεραίων αριθμών)
- Ορίζονται οι απαραίτητες πράξεις:
  - **empty** η άδεια στοίβα
  - **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)

## ◆ Υλοποίηση με πίνακα

```
const size = 100;  
type stack = record  
    arr : array [1..size] of integer;  
    top : 1 .. size+1  
end
```

## ◆ Άδεια στοίβα

```
function empty : stack;  
begin  
    empty.top := 1  
end
```

## ◆ Προσθήκη στοιχείου

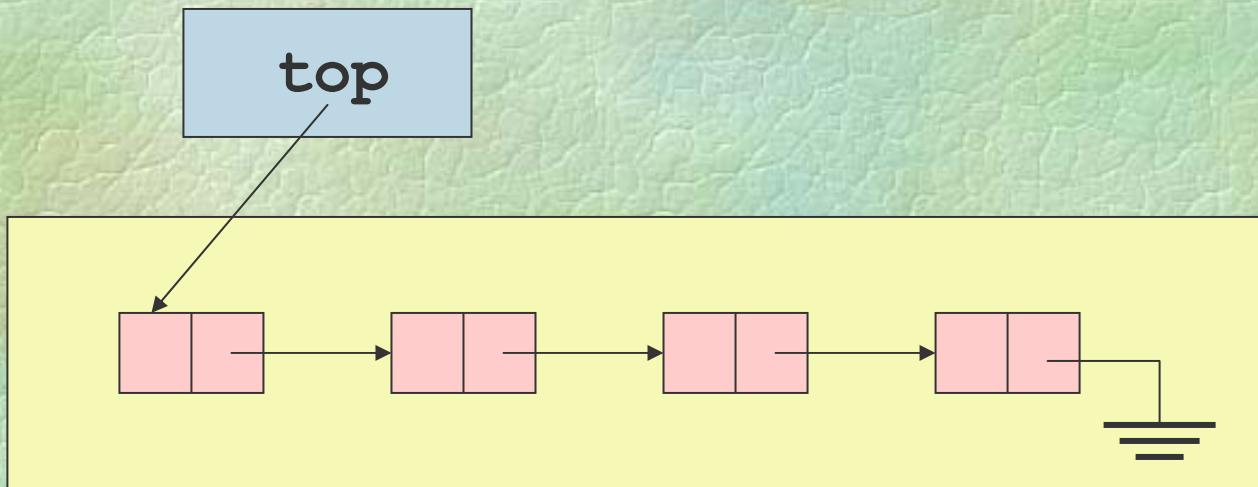
```
procedure push (var s : stack,  
                data : integer);  
begin  
    s.arr[s.top] := data;  
    s.top := s.top + 1  
end
```

## ◆ Αφαίρεση στοιχείου

```
procedure pop (var s : stack;
               var data : integer;
               var nonempty : boolean);

begin
    if s.top <= 1 then
        nonempty := false
    else
        begin
            s.top := s.top - 1;
            data := s.arr[s.top];
            nonempty := true
        end
    end
end
```

- ◆ Υλοποίηση με απλά συνδεδεμένη λίστα



- ◆ Υλοποίηση με απλά συνδεδεμένη λίστα

```
type node = record
    info : integer;
    next : ^node
  end;
stack = ^node
```

- ◆ Άδεια στοίβα

```
function empty : stack;
begin
  empty := nil
end
```

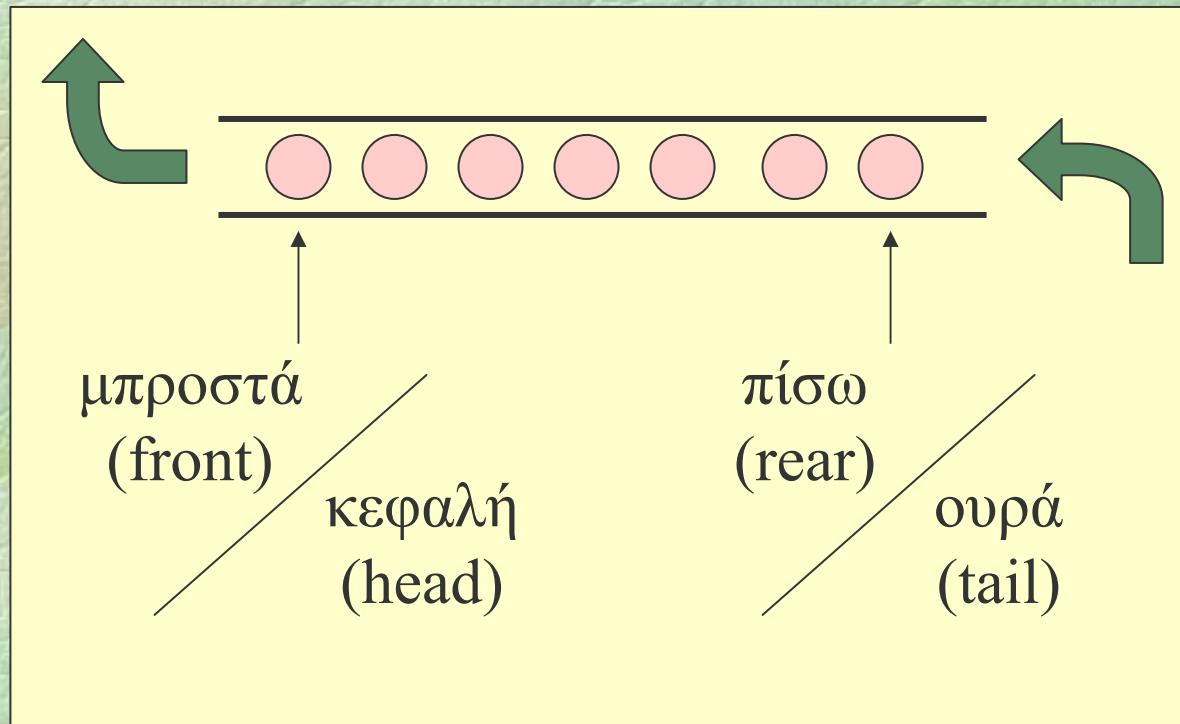
## ◆ Προσθήκη στοιχείου

```
procedure push (var s : stack,  
                data : integer);  
    var p : ^node;  
begin  
    new(p);  
    p^.info := data;  
    p^.next := s;  
    s := p  
end
```

## ◆ Αφαίρεση στοιχείου

```
procedure pop (var s : stack;
               var data : integer;
               var nonempty : boolean);
  var p : ^node;
begin
  if s = nil then
    nonempty := false
  else begin p := s;
           data := s^.info;
           s.top := s^.next;
           dispose(p);
           nonempty := true
  end
end
```

- ◆ 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:

- **empty**

η άδεια ουρά

- **enqueue**

προσθήκη στοιχείου στο τέλος

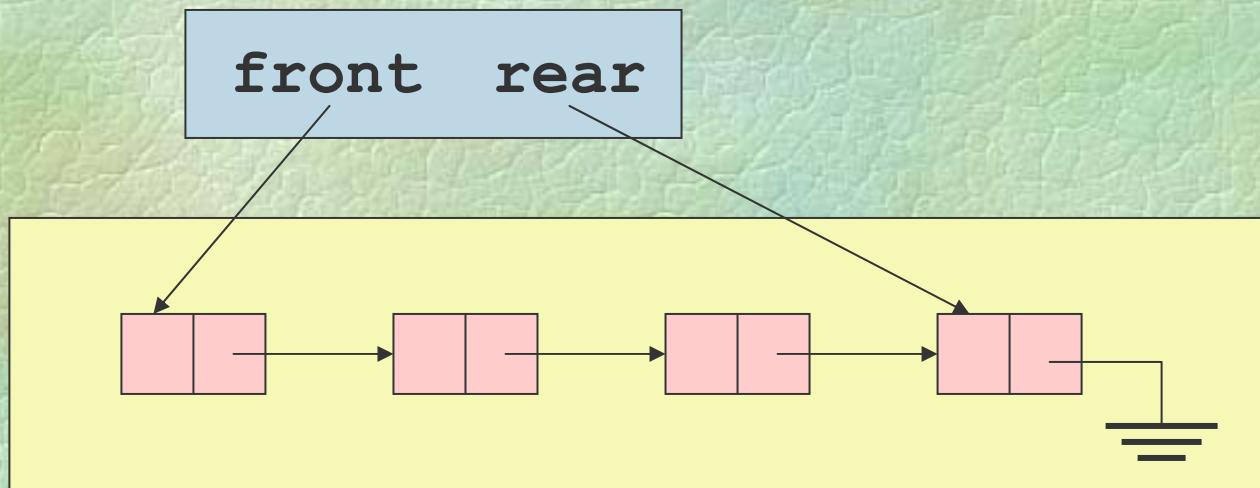
- **dequeue**

αφαίρεση στοιχείου από την αρχή

- **isempty**

έλεγχος για άδεια ουρά

- ◆ Υλοποίηση με απλά συνδεδεμένη λίστα



- ◆ Υλοποίηση με απλά συνδεδεμένη λίστα

```
type node = record
    info : integer;
    next : ^node
  end;
queue = record
    front : ^node;
    rear  : ^node
  end
```

- ◆ Άδεια ουρά

```
function empty : queue;
begin empty.front := nil;
  empty.rear   := nil;
end
```

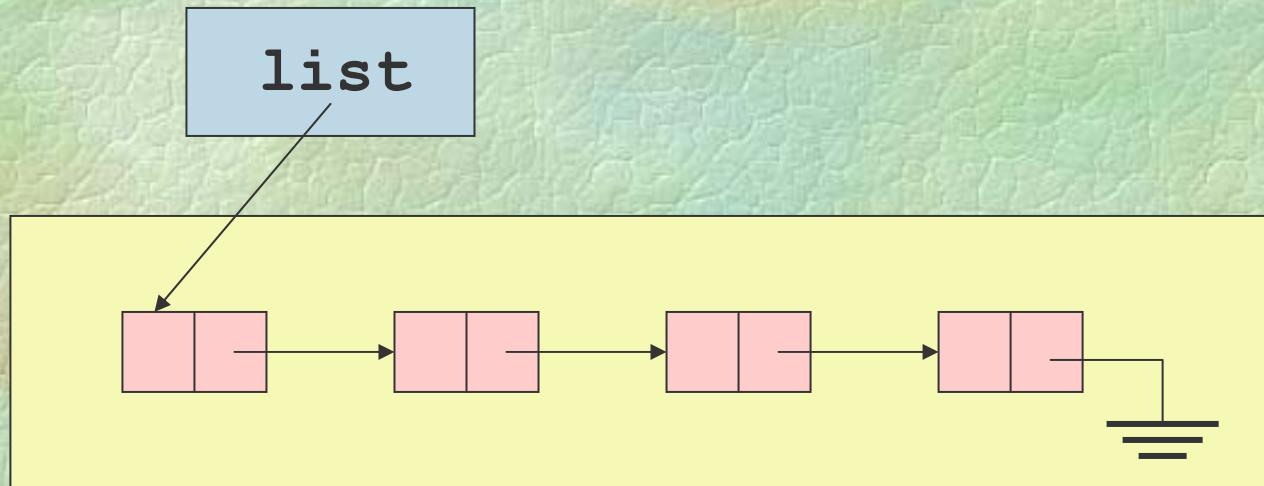
◆ Προσθήκη στοιχείου

```
procedure enqueue (var q : queue,  
                  data : integer) ;  
  var p : ^node;  
begin  
  new(p);  
  p^.info := data;  
  p^.next := nil;  
  if q.front = nil then  
    q.front := p  
  else  
    q.rear^.next := p;  
  q.rear := p  
end
```

◆ Αφαίρεση στοιχείου

```
procedure dequeue (var q : queue;
                   var data : integer;
                   var nonempty : boolean);
  var p : ^node;
begin
  if q.front = nil then
    nonempty := false
  else begin p := q.front;
           data := q.front^.info;
           if q.front = q.rear then
             q.rear := nil
           q.front := q.front^.next;
           dispose(p); nonempty := true
         end
end
```

- ◆ Γενική μορφή απλά συνδεδεμένης λίστας



## ◆ Εισαγωγή στο τέλος

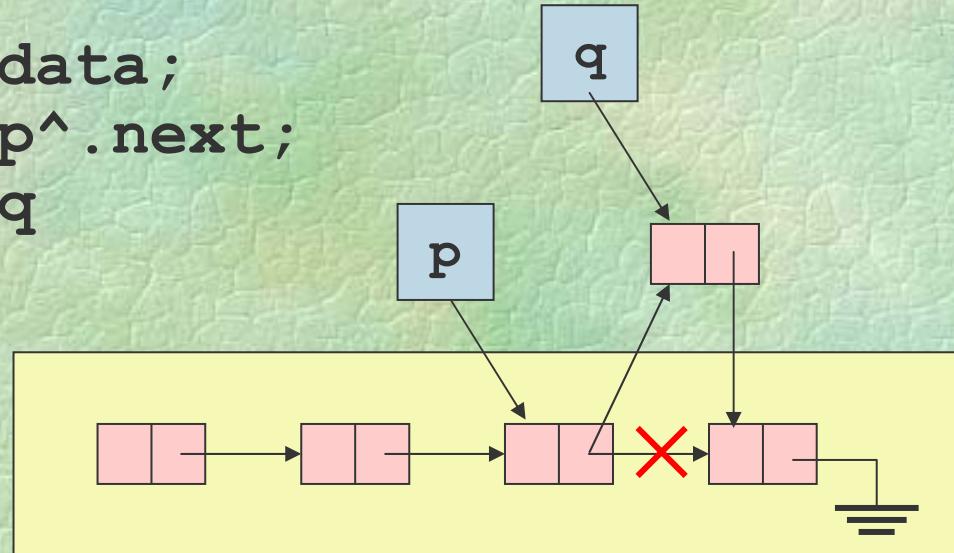
```
procedure insertAtRear (
    var list : ^node;
    data : integer);
    var p, q : ^node;
begin new(p);
    p^.info := data; p^.next := nil;
    if list = nil then list := p
    else begin q := list;
            while q^.next <> nil do
                q := q^.next;
            q^.next := p
        end
    end
```

- ◆ Εισαγωγή μετά τον κόμβο p

```

procedure insertAfter (p : ^node;
                      data : integer) ;
  var q : ^node;
begin
  if p <> nil then
    begin new(q);
      q^.info := data;
      q^.next := p^.next;
      p^.next := q
    end
  end

```

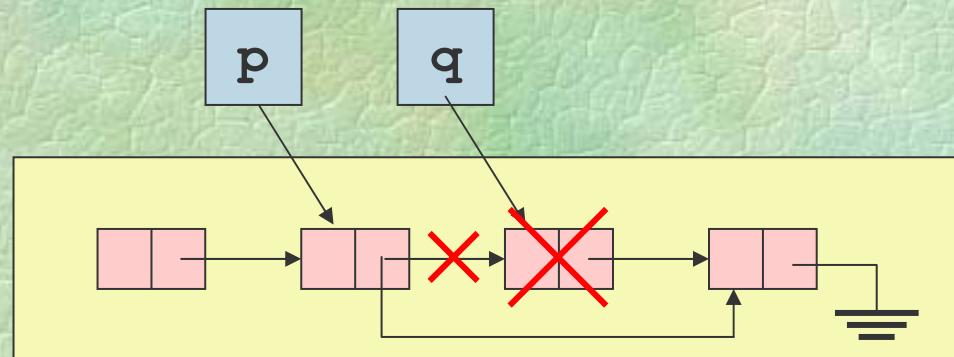


- ◆ Διαγραφή του κόμβου μετά τον p

```

procedure deleteAfter (var p : ^node;
                      var data : integer);
var q : ^node;
begin
  if p<>nil and (p^.next<>nil) then
    begin q := p^.next;
              data := q^.info;
              p^.next := q^.next;
              dispose(q)
    end
end

```



## ◆ Εύρεση στοιχείου

```
function search (list : ^node;
                 data : integer) : ^node;
var p : ^node; found : boolean;
begin
  p := list;
  found := false;
  while (p <> nil) and not found do
    if p^.info = x then
      found := true
    else
      p := p^.next;
  search := p
end
```

## ◆ Αντιστροφή λίστας

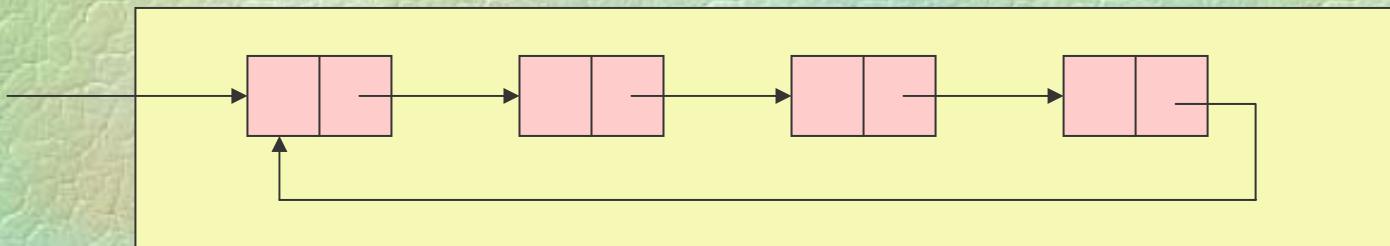
```
procedure reverse (var list : ^node);
    var p, q : ^node;
begin
    q := nil;
    while list <> nil do
begin
    p := list;
    list := p^.next;
    p^.next := q;
    q := p
end;
    list := q
end
```

## ◆ Συνένωση δύο λιστών

```
procedure listconcat (var list1 : ^node;
                      list2 : ^node);
  var p : ^node;
begin
  if list2 <> nil then
    if list1 = nil then list1 := list2
  else begin p := list1;
           while p^.next <> nil do
             p := p^.next;
           p^.next := list2
         end
      end
```

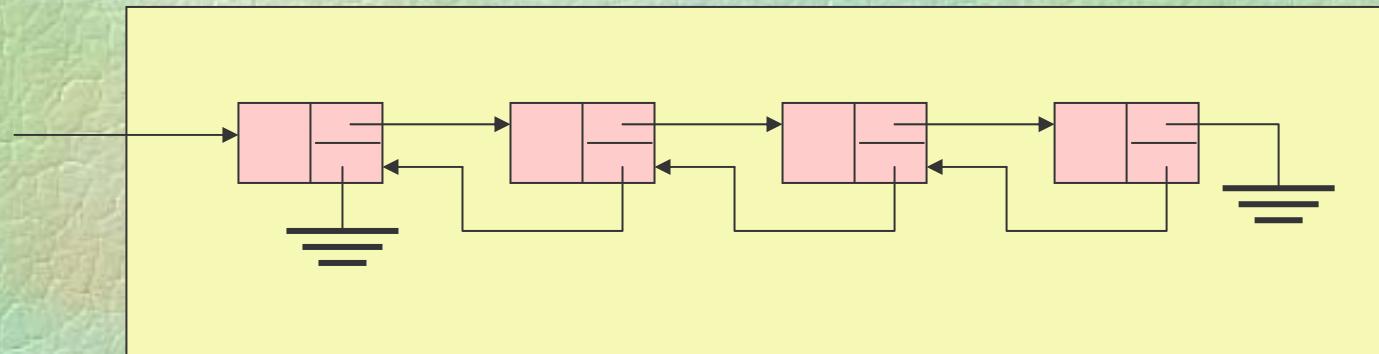
# Κυκλικές λίστες

- ◆ Ο επόμενος του τελευταίου κόμβου είναι πάλι ο πρώτος



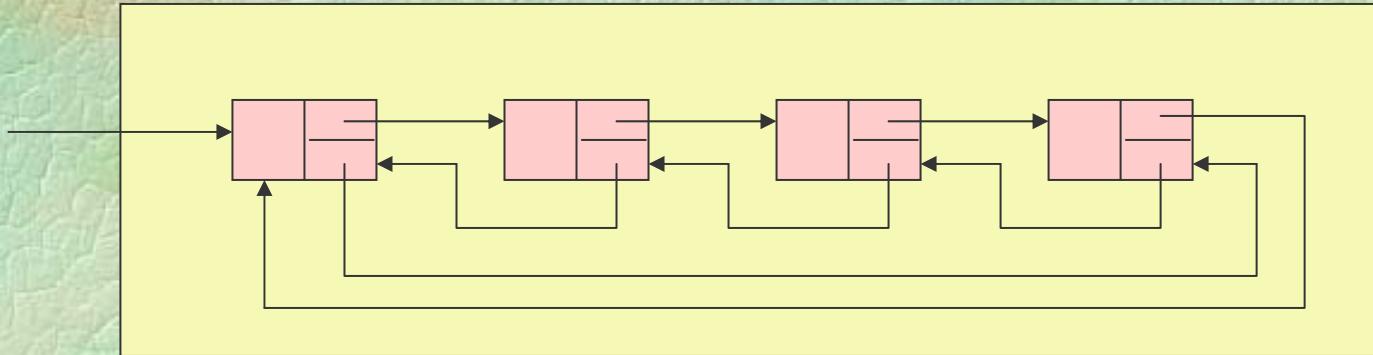
# Διπλά συνδεδεμένες λίστες

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



# Διπλά συνδεδεμένες κυκλικές λίστες

- ◆ Δυο 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) $G = (V, E)$

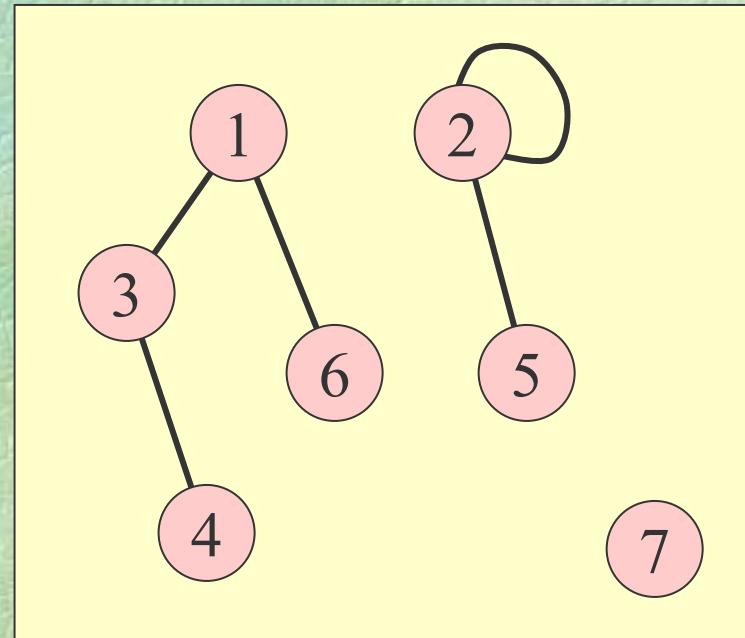
- $V$  Σύνολο κόμβων ή κορυφών
- ΕΣύνολο ακμών, δηλαδή ζευγών κόμβων

## ◆ Παράδειγμα

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

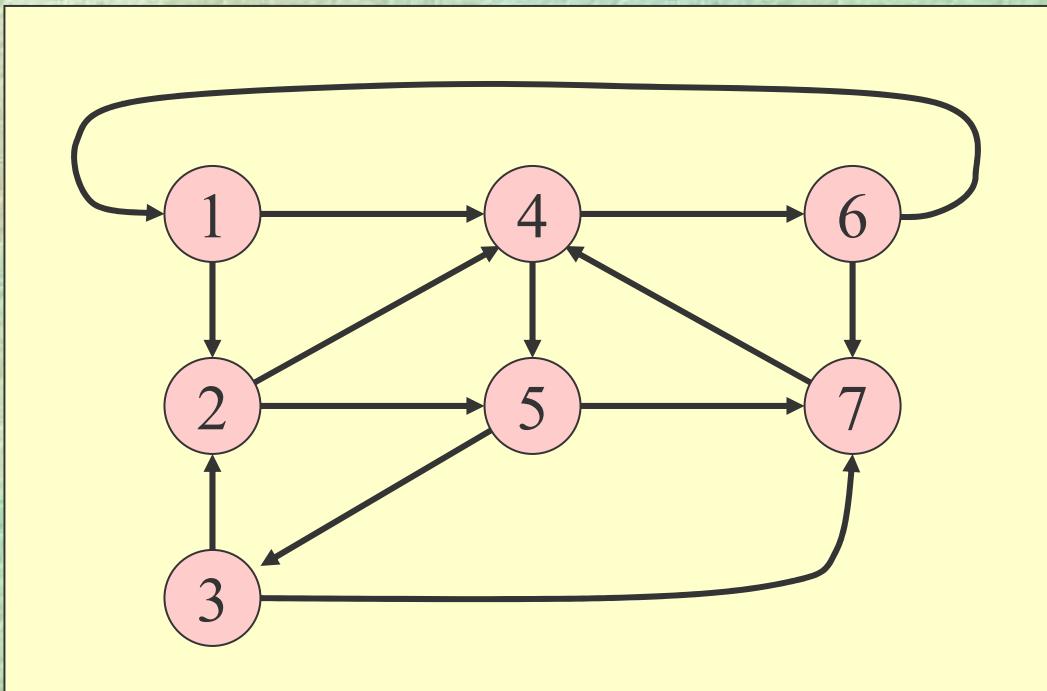
$$E = \{ (x, y) \mid x, y \in V, \\ x+y=4 \text{ ή } x+y=7 \}$$

## ◆ Γραφική παράσταση



## ◆ Κατευθυνόμενος γράφος (directed graph)

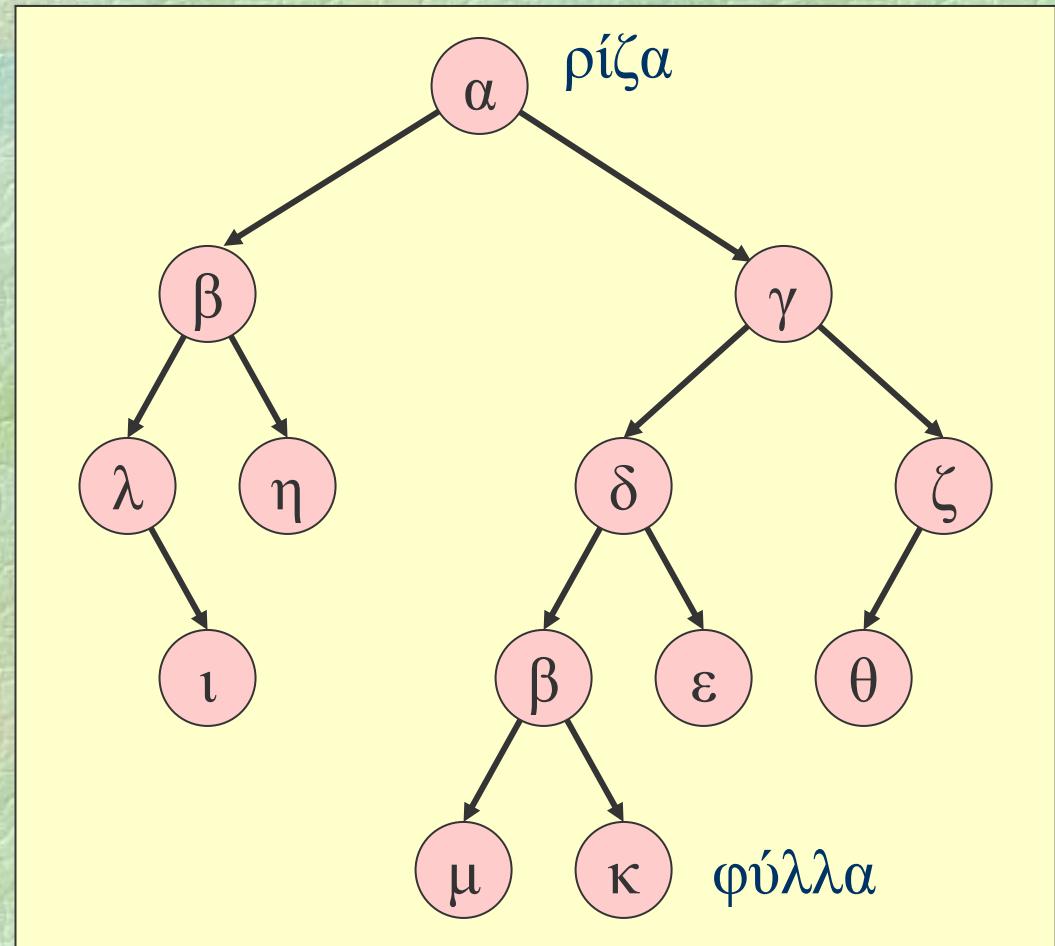
- Οι ακμές είναι διατεταγμένα ζεύγη
- Μπορούν να υλοποιηθούν με δείκτες



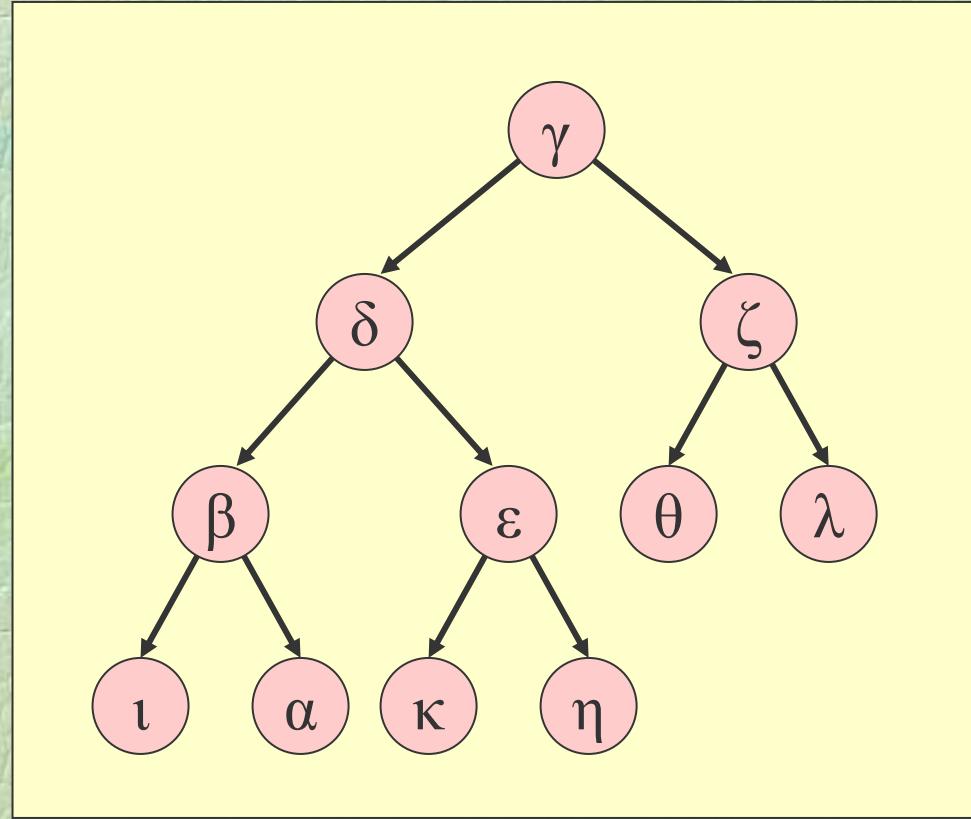
# Δυαδικά δέντρα

(i)

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

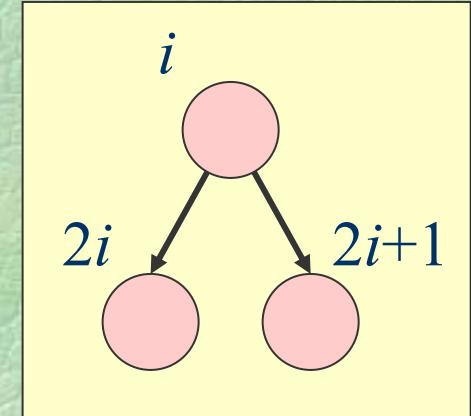


- ◆ Πλήρες δυαδικό δέντρο:
- ◆ Μόνο το κατώτατο επίπεδο μπορεί να μην είναι πλήρες
- ◆ Πλήθος κόμβων =  $n \Rightarrow$  ύψος =  $O(\log n)$



## ◆ Υλοποίηση με πίνακα

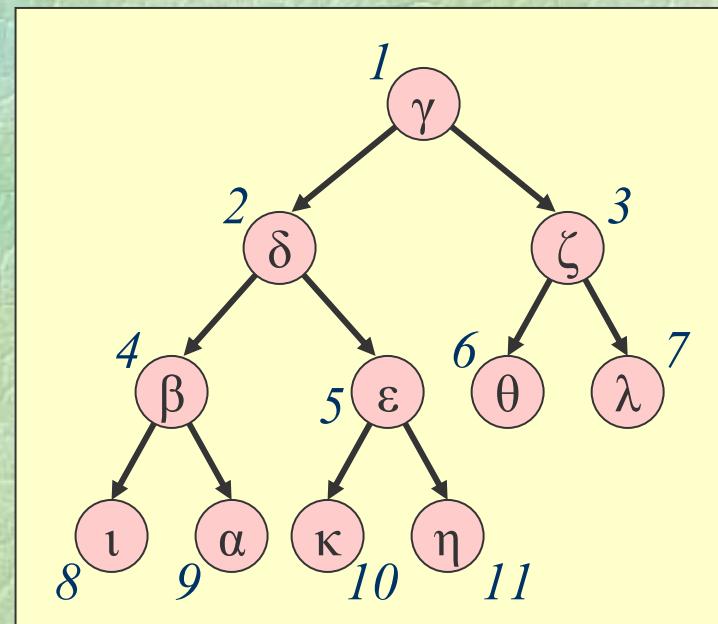
- Αν ένας κόμβος αποθηκεύεται στη θέση  $i$  του πίνακα, τα παιδιά του αποθηκεύονται στις θέσεις  $2i$  και  $2i+1$



## ◆ 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'
  
```

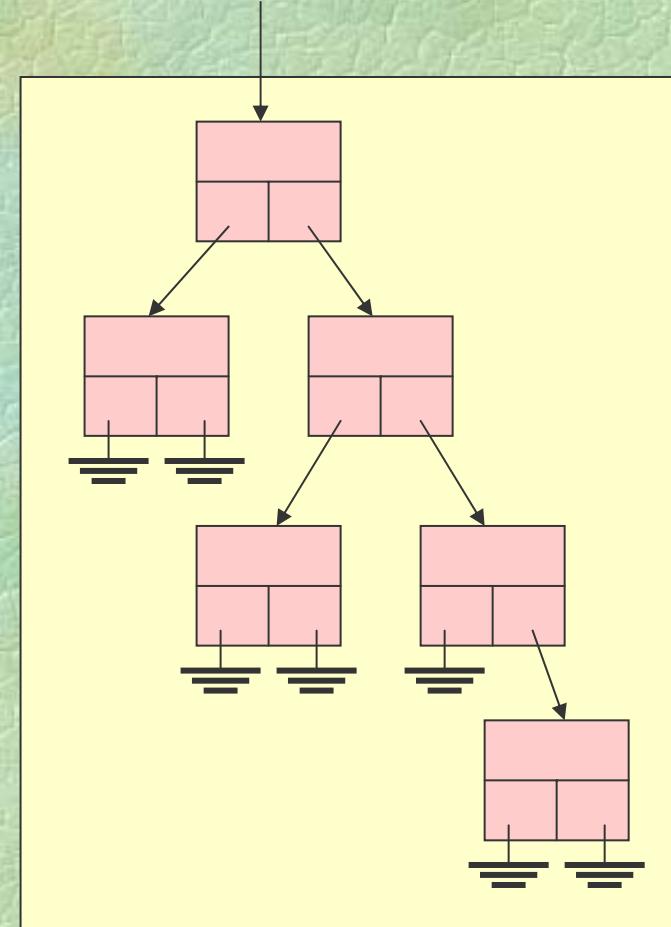
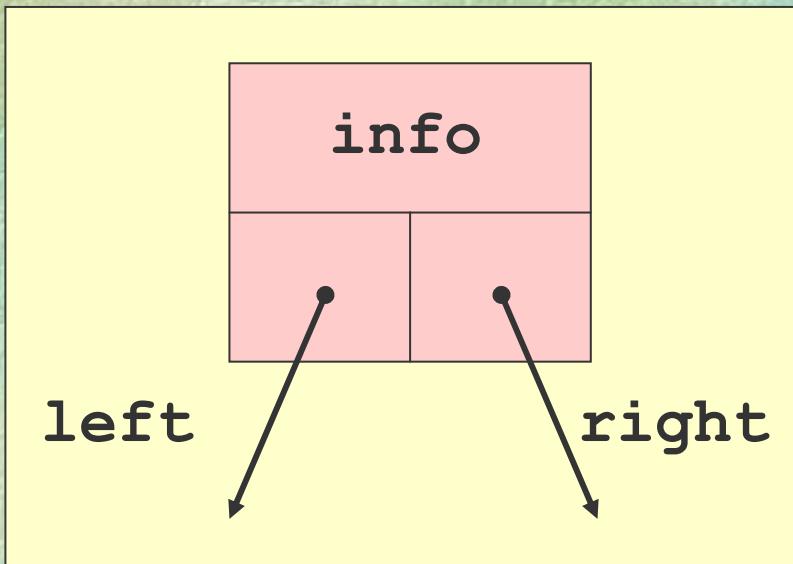


# Δυαδικά δέντρα

(iv)

◆ Υλοποίηση με δείκτες

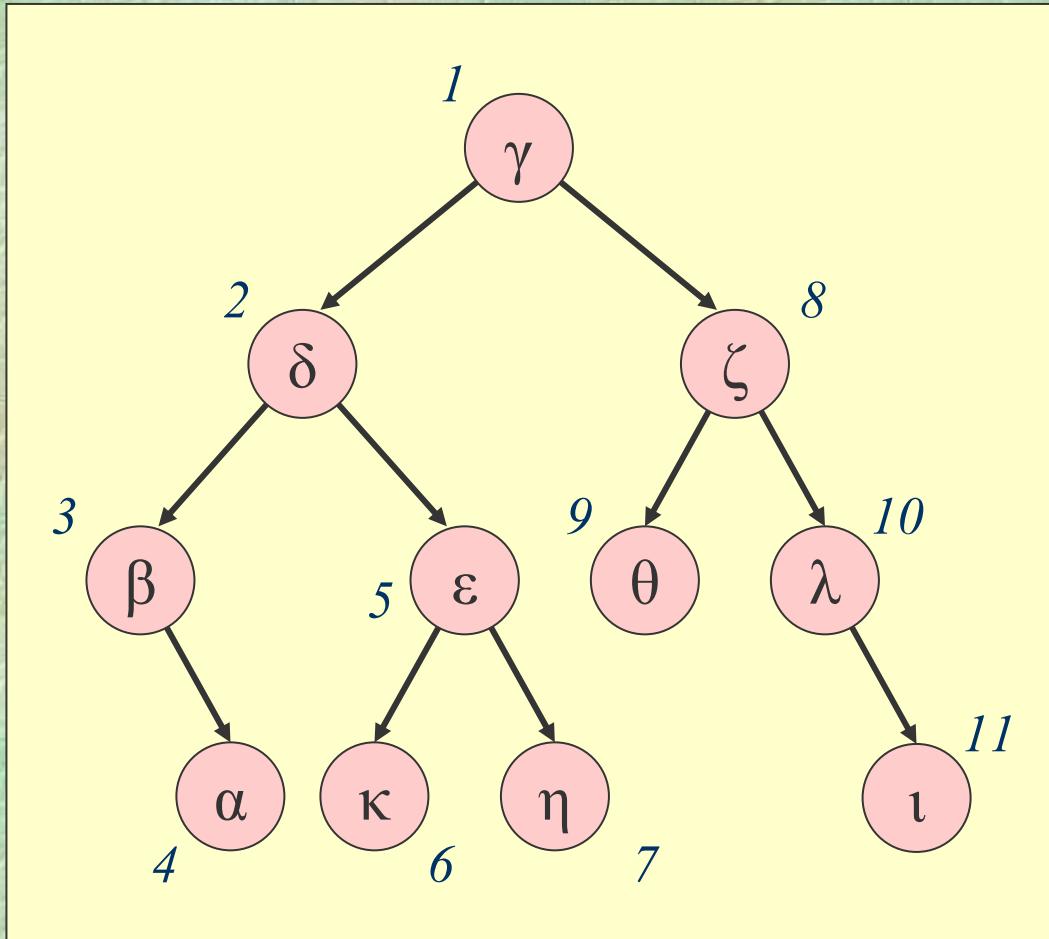
```
type node = record
    info : integer;
    left, right : ^node
end
```



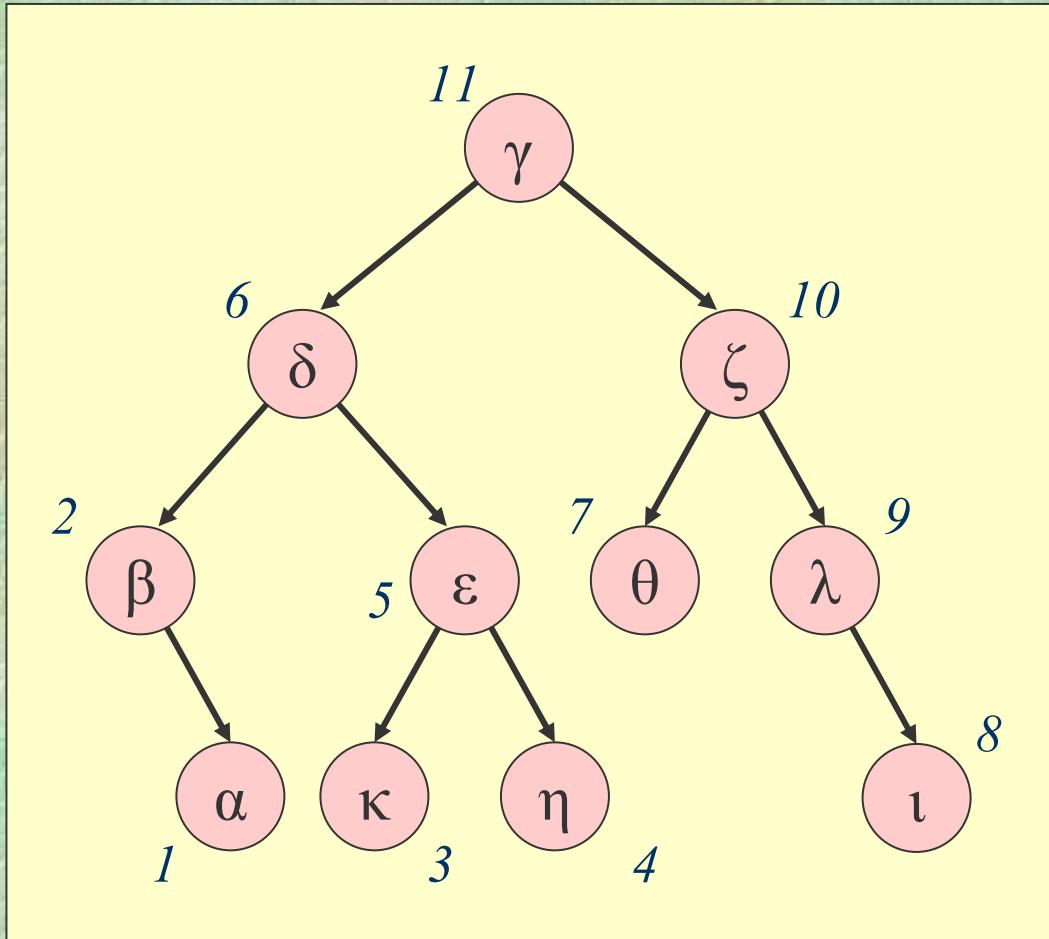
## ◆ Διάσχιση όλων των κόμβων ενός δέντρου

- 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

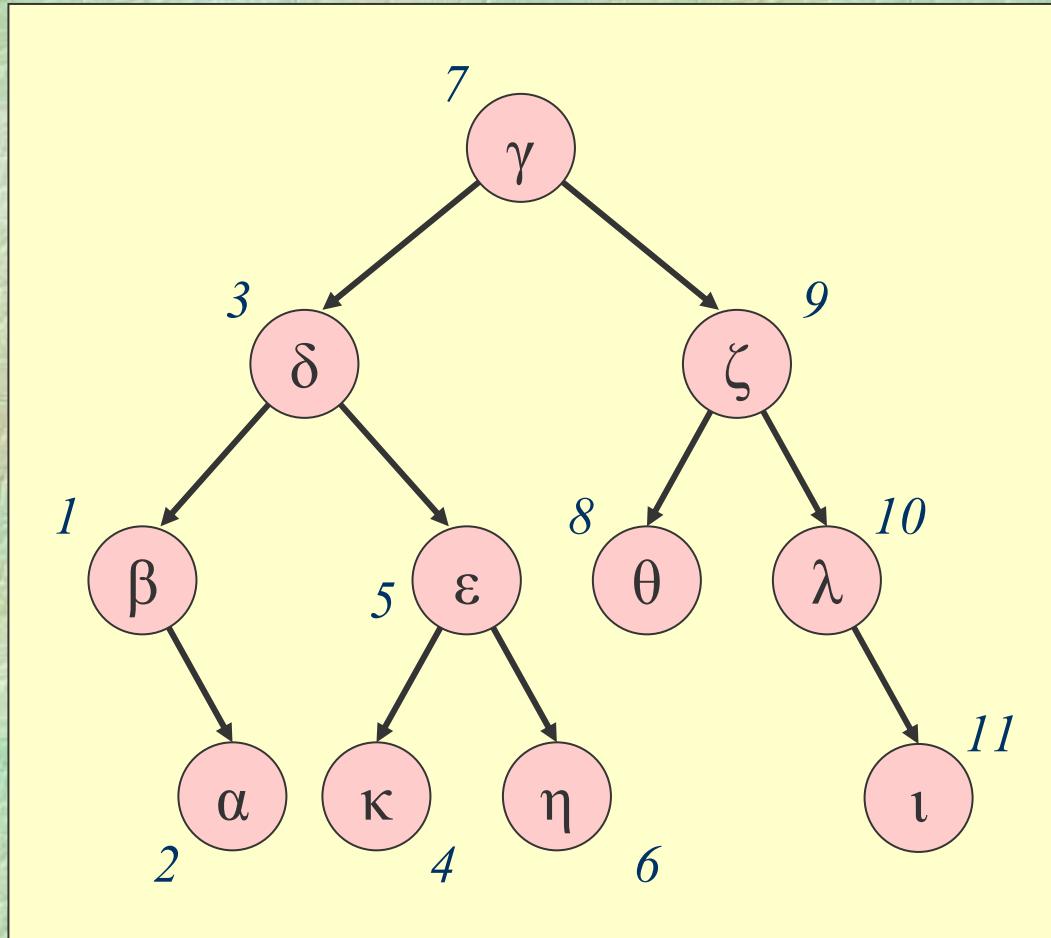
## ◆ Διάσχιση preorder



## ◆ Διάσχιση postorder



## ◆ Διάσχιση inorder



- ◆ Ύλοποίηση της διάσχισης preorder

```
procedure preorder (p : ^node) ;  
begin  
    if p <> nil then  
        begin write(p^.info) ;  
            preorder(p^.left) ;  
            preorder(p^.right)  
        end  
    end
```

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

# 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

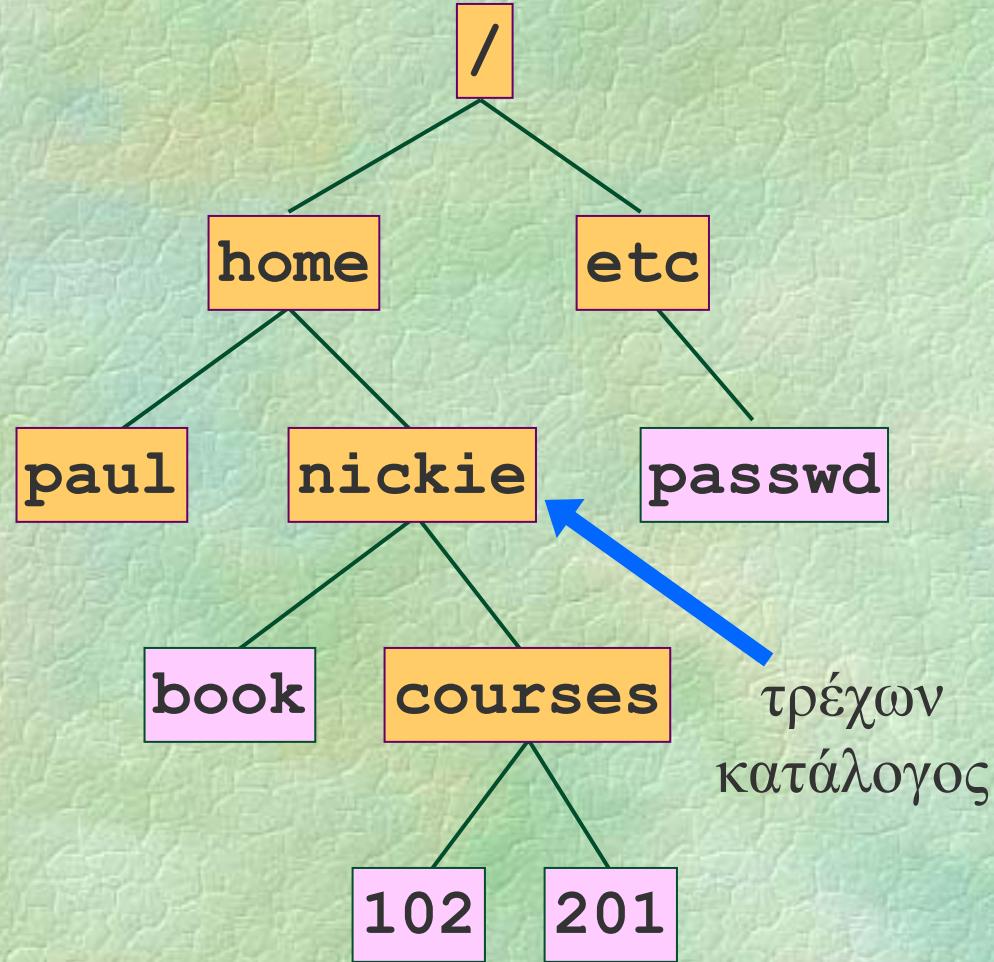
# Το λειτουργικό σύστημα Unix (ii)

## ◆ Απόλυτα ονόματα

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

## ◆ Σχετικά ονόματα

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



## ◆ 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

## ◆ 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*

## ◆ 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 áδειων 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

## ◆ Μετακίνηση μέσα σε αρχείο

$\leftarrow \downarrow \uparrow \rightarrow$	κατά ένα χαρακτήρα
<b>h j k l</b>	(ομοίως)
<b>w</b>	μια λέξη δεξιά
<b>CTRL+F</b>	μια σελίδα μετά
<b>CTRL+B</b>	μια σελίδα πριν
<b>CTRL+D</b>	μισή σελίδα μετά
<b>CTRL+U</b>	μισή σελίδα πριν
<b>0 \$</b>	στην αρχή ή στο τέλος της γραμμής
<b>^</b>	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
<i>n</i>	<b>G</b>	στην <i>n</i> -οστή γραμμή
	<b>G</b>	στην τελευταία γραμμή

# Βασική λειτουργία του vi (iv)

## ◆ Εισαγωγή κειμένου

- i a** 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

## ◆ 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

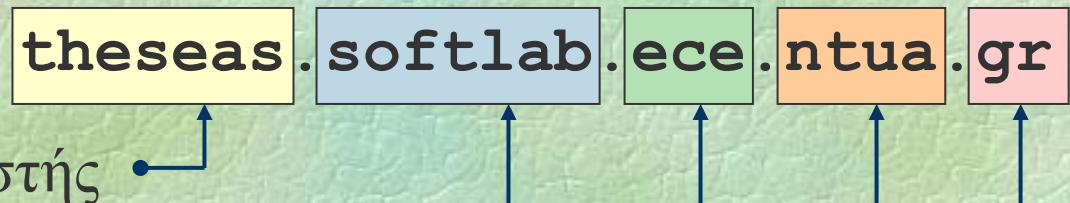
**:q!** έξοδος χωρίς αποθήκευση

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

- Διεύθυνση IP

147.102.1.1

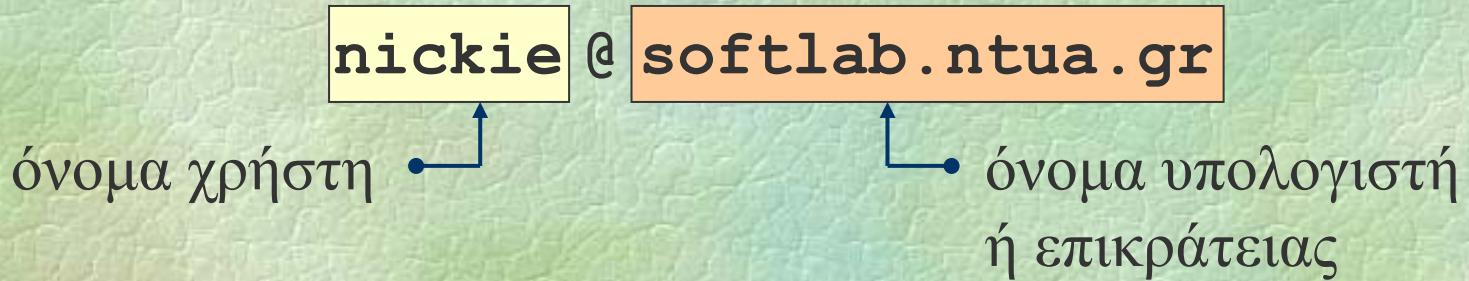
- Όνομα



- 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)

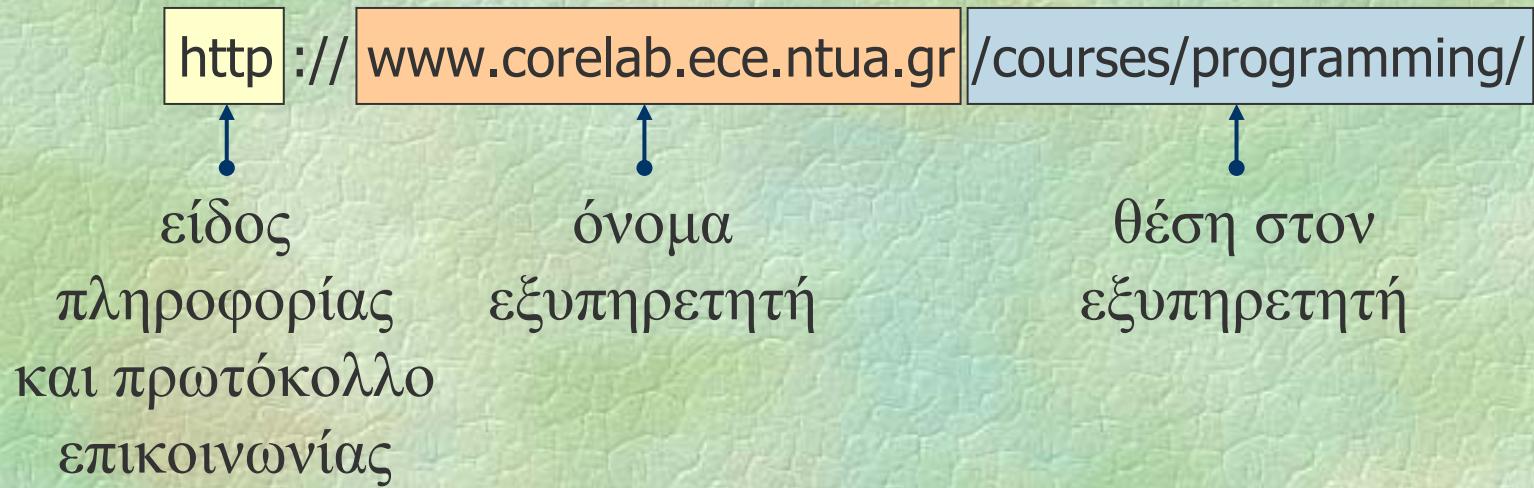
## ◆ Κουτσομπολιό (chat ή IRC)

- **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`