

Διδάσκοντες: Στάθης Ζάχος (zachos@cs.ntua.gr)  
Νίκος Παπασπύρου (nickie@softlab.ntua.gr)  
Δημήτρης Φωτάκης (fotakis@cs.ntua.gr)

### Διαφάνειες παρουσιάσεων

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

21/10/11

## Εισαγωγή

(i)

### ◆ Σκοπός του μαθήματος

- Εισαγωγή στην **πληροφορική** (computer science)
- Εισαγωγή στον **προγραμματισμό** ηλεκτρονικών υπολογιστών (H/Y)
- Μεθοδολογία **αλγοριθμικής επίλυσης** προβλημάτων

## Εισαγωγή

(ii)

### ◆ 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**

## Εισαγωγή

(iii)

### ◆ Φυσική γλώσσα

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

### ◆ Τυπική γλώσσα

- **Αυστηρότατη** σύνταξη και σημασιολογία

### ◆ Γλώσσα προγραμματισμού

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

## Εισαγωγή

(iv)

### ◆ Πληροφορική

Ηλεκτρονικοί  
υπολογιστές  
(engineering)

Σχεδίαση και  
κατασκευή

Μαθηματικά

Θεωρία και  
αναλυτική μέθοδος

### ◆ Κεντρική έννοια:

**υπολογισμός (computation)**

## Εισαγωγή

(v)

### ◆ 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.



## Εισαγωγή

(vi)

### ◆ 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 ανουσιών 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  $(\alpha + \beta)(\alpha - \beta) = \alpha^2 - \beta^2$

## Εισαγωγή

(vii)

### ◆ 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

## Εισαγωγή

(viii)

### ◆ Τριτοβάθμια εκπαίδευση

- Η τεχνολογία αλλάζει αέναα και γρήγορα – τα θεμέλια μένουν
- Αυτά τα θεμέλια πρέπει να είναι η ραχοκοκαλιά στην τριτοβάθμια εκπαίδευση: έμφαση στην αλγοριθμική 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

## Εισαγωγή

(ix)

### ◆ Να μην ξεχνάμε ότι

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

## Εισαγωγή

(x)

### ◆ Κατασκευή υπολογιστικών μηχανών

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

## Εισαγωγή

(xi)

### ◆ Κατασκευή υπολογιστών

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



## Εισαγωγή (xii)

### ◆ Κατασκευή υπολογιστών

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

## Εισαγωγή (xiii)

### ◆ 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*



## Εισαγωγή (xiv)

### ◆ Θεμέλια της πληροφορικής

- Μαθηματική λογική
- **Αριστοτέλης**: συλλογισμοί

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

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

## Εισαγωγή (xv)

### ◆ Πρόγραμμα του Leibniz:

#### θεμελίωση των μαθηματικών

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

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

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

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

## Εισαγωγή (xvi)

### ◆ Θεωρία

- Συνολοθεωρία, **Cantor, Frege**  $\in$
- Παράδοξο του **Russel**

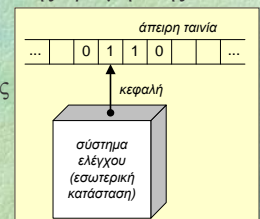
$$A = \{ x \mid x \notin x \} \quad \begin{array}{l} A \in A \rightarrow A \notin A \\ A \notin A \rightarrow A \in A \end{array}$$

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

## Εισαγωγή (xvii)

### ◆ 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, υπολογισσιμότητα





## Εισαγωγή (xviii)

### ◆ Μη πληρότητα (incompleteness)

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

## Εισαγωγή (xix)

### ◆ 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

## Εισαγωγή (xx)

### ◆ Υπολογιστής

- επεξεργαστής
- μνήμη
- συσκευές εισόδου/εξόδου

### ◆ Ιδιότητες

- αυτόματο χωρίς εξυπνάδα
- μεγάλη ταχύτητα
- ακρίβεια στις πράξεις

## Γλώσσες προγραμματισμού (i)

### ◆ Γλώσσα μηχανής

0110110      11011011  
διεύθυνση      εντολή

### ◆ Συμβολική γλώσσα (assembly)

label:      add    ax, bx  
διεύθυνση      πράξη    δεδομένα

### ◆ Γλώσσες χαμηλού και υψηλού επιπέδου

### ◆ Υλοποίηση γλωσσών προγραμματισμού

- μεταγλωττιστής (compiler)
- διερμηνέας (interpreter)

## Γλώσσες προγραμματισμού (ii)

### ◆ Κυριότερες γλώσσες, ιστορικά

1950	FORTRAN, LISP, COBOL, Algol,
1960	BASIC, PL/I
1970	Pascal, C, Smalltalk, Prolog, ML, Logo
1980	C++, Modula-2, Ada, Perl
1990	Java, Python, Ruby, Haskell, PHP
2000	C#, ...

## Γλώσσες προγραμματισμού (iii)

### ◆ Pascal

- Niklaus Wirth (1971)
- Γλώσσα γενικού σκοπού (general purpose)
- 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 (C)

(i)

```
#include <stdio.h>  
void main ()  
{  
  printf("hello world\n");  
}
```

```
#include <stdio.h>  
void main ()  
{  
  printf("hello " "world\n");  
}
```

```
#include <stdio.h>  
void main ()  
{  
  printf("hel  
  printf("wor  
}
```

```
#include <stdio.h>  
void main ()  
{  
  printf("hello world");  
  printf("\n");  
}
```

## 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.
```

## Ασκήσεις (C)

(ii)

```
#include <stdio.h>  
  
void hello()  
{  
  printf("h  
}  
  
void main()  
{  
  hello();  
}
```

```
#include <stdio.h>  
void hello()  
{  
  printf("hello world\n");  
}  
  
void main()  
{  
  int i;  
  for (i=0; i<20; i++)  
    hello();  
}
```

## Ασκήσεις

(iii)

```
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.
```

## Ασκήσεις (C)

(iii)

```
#include <stdio.h>  
const int n=20;  
int i;  
  
void num_hello()  
{  
  printf("%d hello world\n", i);  
}  
  
void main()  
{  
  for (i=0; i<n; i++)  
    num_hello();  
}
```



## Ασκήσεις

(iv)

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

## Ασκήσεις (C)

(iv)

```
#include <stdio.h>
void hello()
{
  printf("hello world\n")
}
void main()
{
  int i,n;
  printf("Give number of greetings"
        "then press <enter>:");
  scanf("%d\n", &n);
  for (i=0; i<n; i++) hello();
}
```

## Ασκήσεις

(v)

```
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.
```

## Ασκήσεις (C)

(v)

```
#include <stdio.h>
void hello()
{
  printf("hello world\n");
}
void main()
{
  int i,n;
  printf("Give number of greetings"
        "then press <enter>");
  scanf("%d\n", &n);
  if (n<0)
    printf("#is negative\n");
  else
    for (i=0;i<n;i++) hello();
}
```

## Δομή του προγράμματος

(i)

```
program example(input, output);
var i, j : integer;
begin
  i:=15; j:=23;
  write('sum of i and j is: ');
  i:=i+j;
  write(i)
end.
```

*επικεφαλίδα*

*δηλώσεις*

*κυρίως σώμα*

## Δομή του προγράμματος (C)

(i)

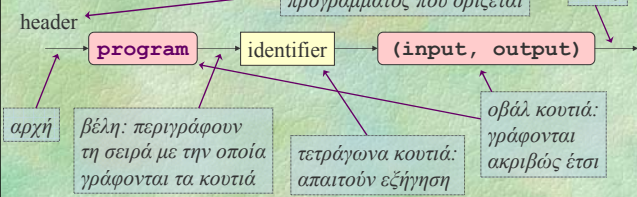
```
#include <stdio.h>
int i, j;
void main()
{
  i=15; j=23;
  printf("sum of i and j is: ");
  i=i+j;
  printf("%d", i);
}
```

## Δομή του προγράμματος (ii)

### ◆ Επικεφαλίδα

το όνομα του τμήματος προγράμματος που ορίζεται

τέλος



### Συντακτικό διάγραμμα

- περιγράφει τη σύνταξη ενός τμήματος του προγράμματος

## Δομή του προγράμματος (iii)

### ◆ 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;
```

## Δομή του προγράμματος (iv)

### ◆ Απλοί τύποι μεταβλητών

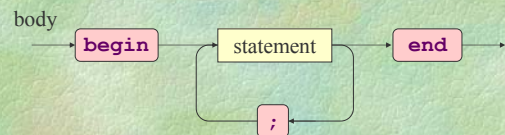
- **integer** ακέραιοι αριθμοί 0, 1, -3
- **real** πραγματικοί αριθμοί 3.14
- **char** χαρακτήρες 'a'
- **boolean** λογικές τιμές true, false

### ◆ Δήλωση περισσότερων μεταβλητών

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

## Δομή του προγράμματος (v)

### ◆ Κυρίως σώμα



### ◆ Σχόλια

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

## Δομή του προγράμματος (vi)

### ◆ 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

## Δομή του προγράμματος (C) (vi)

### ◆ Υποπρογράμματα

```
#include <stdio.h>
```

```
int i, j;
void add();
{
    i=i+j;
}

void main()
{
    i=15; j=23;
    printf("sum of i and j is: ");
    add();
    printf("%d", i);
}
```



## 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;**

## Τί σημαίνει ορθό πρόγραμμα (ii)

### ◆ 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** διαίρεση με το μηδέν

## Τί σημαίνει ορθό πρόγραμμα (iv)

### ◆ Ο μεταγλωττιστής μπορεί να εντοπίσει σε ένα πρόγραμμα την ύπαρξη

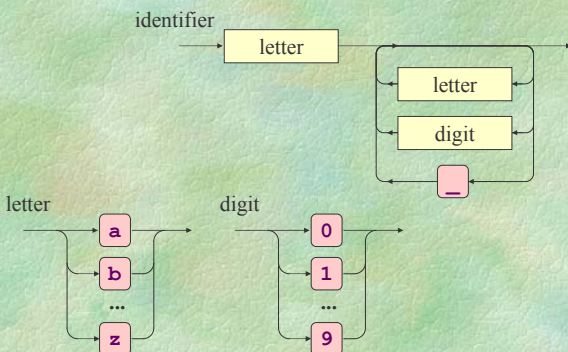
- **συντακτικών** σφαλμάτων
- **νοηματικών** σφαλμάτων

### ◆ 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

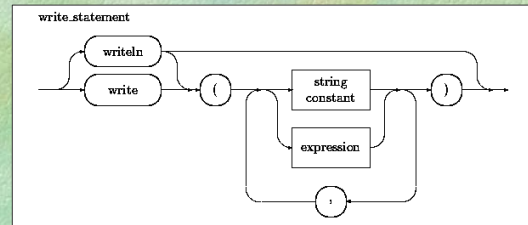
```
write('Hello world')
write(x)
write(n+1)
write(x, y)
write('Η τιμή του x είναι ', x)
```

### ◆ Έξοδος με αλλαγή γραμμής

```
write('Η τιμή του');
writeln(' x είναι ', x);
writeln('Η τιμή του y είναι ', y)
```

## Επικοινωνία με το χρήστη (ii)

### ◆ Συντακτικό διάγραμμα



## Επικοινωνία με το χρήστη (iii)

### ◆ Είσοδος από το πληκτρολόγιο

```
read(a)
```

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

```
readln(b)
readln(x, y)
```

## Επικοινωνία με το χρήστη (iv)

### ◆ 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.
```

## Επικοινωνία με το χρήστη (C) (iv)

### ◆ Παράδειγμα

```
#include <stdio.h>
void main()
{
int n, m, sum;
printf("Προσθέτω δυο ακεραίους\n");
printf("Δώσε το n: ");
scanf("%d\n", &n);
printf("Δώσε το m: ");
scanf("%d\n", &m);
sum = n + m;
printf("Το άθροισμα %d + %d είναι: ", n, m);
printf("%d\n", sum);
}
```

## Αριθμητικές παραστάσεις (i)

### ◆ Απλές παραστάσεις

- σταθερές και μεταβλητές

### ◆ Απλές πράξεις

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



## Αριθμητικές παραστάσεις (ii)

- ◆ 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.  $(x+1)*(y-1)$

## Λογικές παραστάσεις (i)

- ◆ Συγκρίσεις
  - ισότητα, ανισότητα =, <>
  - μεγαλύτερο, μικρότερο >, <
  - μεγαλύτερο ή ίσο, μικρότερο ή ίσο >=, <=
- ◆ Λογικές πράξεις
  - σύζευξη (και) **and**
  - διάζευξη (ή) **or**
  - άρνηση (όχι) **not**

## Λογικές παραστάσεις (ii)

- ◆ Πίνακες αλήθειας λογικών πράξεων

p	q	p and q	p	q	p or q
false	false	false	false	false	false
false	true	false	false	true	true
true	false	false	true	false	true
true	true	true	true	true	true

p	not p
false	true
true	false

## Λογικές παραστάσεις (iii)

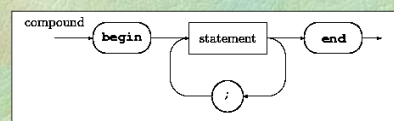
- ◆ Προτεραιότητα λογικών τελεστών
  - **not** : μεγαλύτερη προτεραιότητα από όλους
  - **and** : όπως ο πολλαπλασιασμός
  - **or** : όπως η πρόσθεση
- π.χ. **not p and q or r**  
 $\equiv ((\text{not } p) \text{ and } q) \text{ or } r$
- π.χ. **x>3 and not y=5** **Λάθος!**  
 $\equiv x > (3 \text{ and } (\text{not } y)) = 5$
- 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 συνδυασμό με άλλες δομές
- ◆ Συντακτικό διάγραμμα





## Σύνθετη εντολή

(ii)

### ◆ 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
```

## Σύνθετη εντολή (C)

(ii)

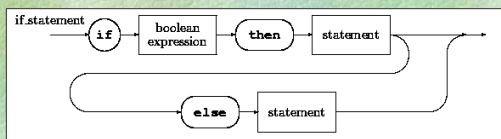
### ◆ Παραδείγματα

```
{
  x=2; y=3; z=3;
  printf("%d %d %d\n",x,y,z);
}
{
  x=2; y=3;
  {
    z=3;
    printf("%d %d %d",x,y,z);
  }
  printf("\n");
}
```

## 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 (C)

(ii)

### ◆ Παραδείγματα

```
if (x>10)
  x = x+1;

if (age<10)
  printf("παιδί");

if ((year>1900) && (year<=2000))
  printf("20ός αιώνας");

if ((year%4==0) &&
  (year%100!=0) ||
  (year%400==0) &&
  (year%4000!=0))
  printf("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 (C)

(iii)

### ◆ Παραδείγματα (συνέχεια)

```
if (changed)
{
    printf("Το αρχείο άλλαξε\n");
    changed = 0;
}

if (x&2==0) printf("άριτος");
else printf("περιττός");

if (mine) { me=1; you=0; }
else { me=0; you=1; }

if (x>y) printf("μεγαλύτερο");
else if (x<y) printf("μικρότερο");
else printf("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('άξονας των x')
```

## Εντολή if (C)

(iv)

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

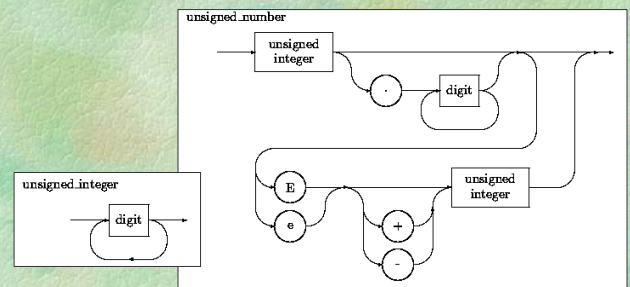
### ◆ Παράδειγμα

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

## Σύνταξη παραστάσεων

(i)

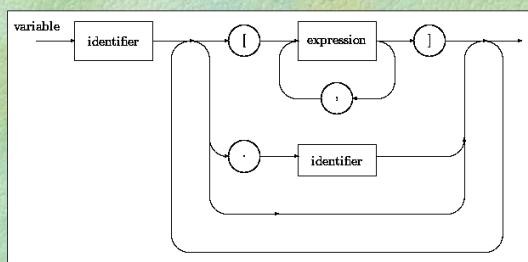
### ◆ Σταθερές



## Σύνταξη παραστάσεων

(ii)

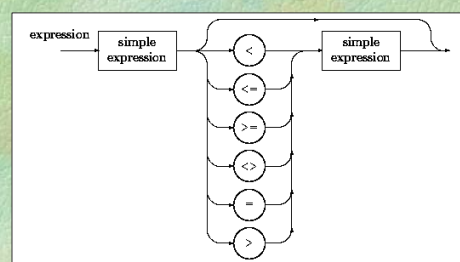
### ◆ Μεταβλητές



## Σύνταξη παραστάσεων

(iii)

### ◆ Παράσταση

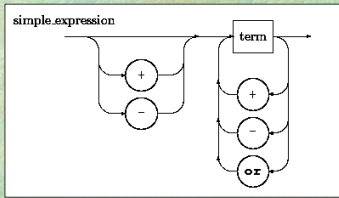




## Σύνταξη παραστάσεων

(iv)

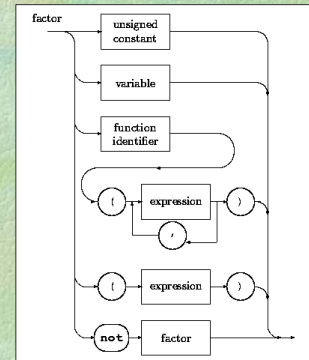
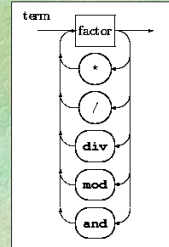
### ◆ Απλή παράσταση



## Σύνταξη παραστάσεων

(v)

### ◆ Όροι και παράγοντες



## Παραδείγματα παραστάσεων

(i)

Συντακτικά ορθή παράσταση	Πώς διαβάζεται (σχόλια)	Τύπος αποτελέσματος	Τιμή αποτελέσματος
$5 = 1$	5 ίσον με 1	boolean	false
$5 >= 1$	5 μεγαλύτερο ή ίσον με το 1	boolean	true
$5 <> 1$	5 διάφορο από το 1	boolean	true
$grade > small$	grade μεγαλύτερο του small	boolean	?
<b>not done</b>	άρνηση του done	boolean	?
$(5 <> 1) \text{ and } (1 > 2)$	Χρήση του τελεστή <b>and</b>	boolean	false

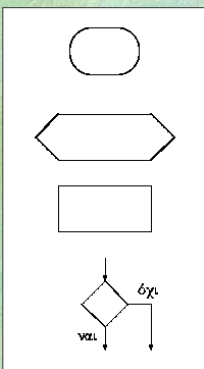
## Παραδείγματα παραστάσεων

(ii)

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

## Λογικά διαγράμματα ροής

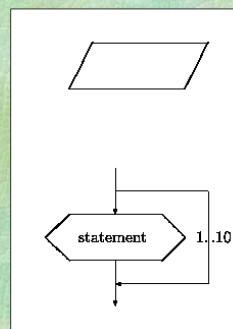
(i)



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

## Λογικά διαγράμματα ροής

(ii)



- ◆ Λειτουργία εισόδου/εξόδου
- ◆ Επανάληψη (βρόχος)



## Εντολή case (i)

(i)

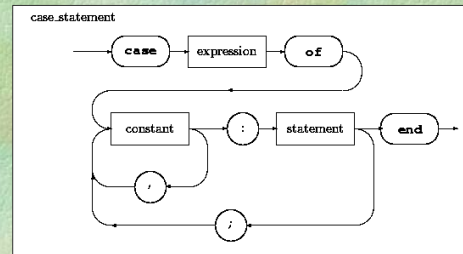
- ◆ Εκτέλεση υπό συνθήκη για πολλές διαφορετικές περιπτώσεις
- ◆ Προσφέρεται π.χ. αντί του:

```
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')
```

## Εντολή case (ii)

(ii)

- ◆ Συντακτικό διάγραμμα



## Εντολή case (iii)

(iii)

- ◆ 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 case (C) (iii)

(iii)

- ◆ Παραδείγματα

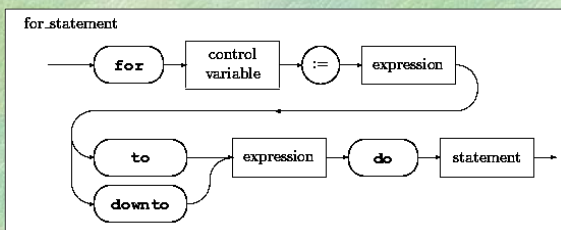
```
switch (month)
{
  case 1: printf("Ιανουάριος"); break;
  case 2: printf("Φεβρουάριος"); break;
  case 3: printf("Μάρτιος"); break;
  ...
  case 12: printf("Δεκέμβριος"); break;
}

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

## Εντολή for (i)

(i)

- ◆ Βρόχος με σταθερό αριθμό επαναλήψεων
- ◆ Συντακτικό διάγραμμα

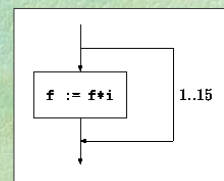


## Εντολή for (ii)

(ii)

- ◆ 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
```





## Εντολή for (C)

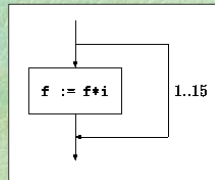
(ii)

### ◆ Παραδείγματα

```
for (i=1; i<=10; i++) printf("%d\n",i);  
for (i=10; i>=1; i--) printf("%d\n",i);
```

```
for (i=41; i>=-3; i--)  
    printf("*");
```

```
f=1;  
for (i=1; i<=15; i++) f=f*i;
```



## Εντολή for

(iii)

### ◆ 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
```

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

## Εντολή for (C)

(iii)

### ◆ Παραδείγματα (συνέχεια)

```
for (i=1; i<=5; i++)  
{  
    for (j=1; j<=10; j++)  
        printf("*");  
    printf("\n");  
}
```

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

```
for (i=1; i<=5; i++)  
{  
    for (j=1; j<=2*i; j++)  
        printf("*");  
    printf("\n");  
}
```

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

## Εντολή for

(iv)

### ◆ 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

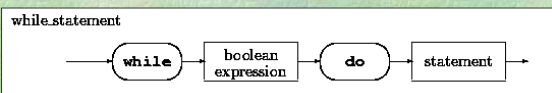
### ◆ Τα όρια υπολογίζονται μια φορά στην αρχή

## Εντολή while

(i)

### ◆ Βρόχος όσο ικανοποιείται μια συνθήκη

### ◆ Συντακτικό διάγραμμα



### ◆ Παραδείγματα

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

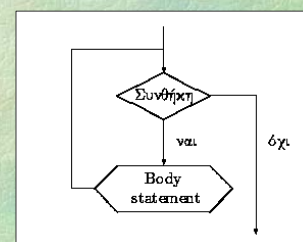
```
while state and (x>15) do  
begin x:=x-5; write(x) end
```

## Εντολή while

(ii)

### ◆ Διάγραμμα ροής

```
while συνθήκη do σώμα
```



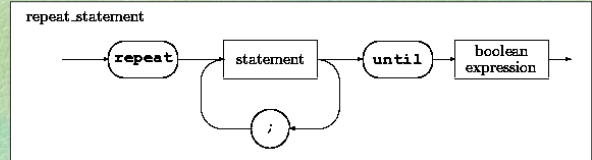


## Εντολή while (iii)

- ◆ 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

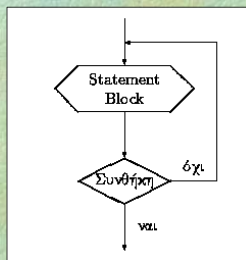


- ◆ Παράδειγμα

```
x:=1;
repeat writeln(x) ; x:=2*x until x>=500
```

## Εντολή repeat (ii)

- ◆ Διάγραμμα ροής



## Εντολή repeat (iii)

- ◆ 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)

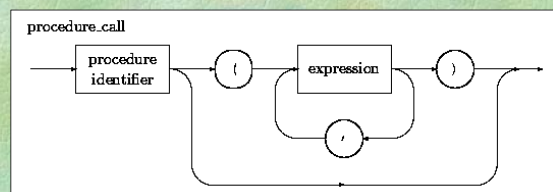
- ◆ Παράδειγμα

```
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.
```

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

## Κλήση διαδικασίας

- ◆ Συντακτικό διάγραμμα



- ◆ Παραδείγματα

```
myproc(x, y+1, 3)
write(x+y, 'wednesday', x-y); writeln
```

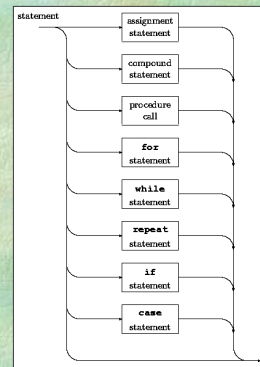


## Κενή εντολή

- ◆ 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



## Δομημένος προγραμματισμός (i)

- ◆ **Ιδέα:** κάθε ανεξάρτητη λειτουργία του προγράμματος πρέπει να αντιστοιχεί σε ανεξάρτητο υποπρόγραμμα
- ◆ Πλεονεκτήματα
  - Ευκολότερη ανάπτυξη προγραμμάτων («διαίρει 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 := sqrt(x)+3*y+5
  end;
begin
  write('Δώσε το n1: '); readln(n1);
  write('Δώσε το n2: '); readln(n2);
  arith(n1, n2, sum, expr);
  writeln('Άθροισμα = ', sum,
         ' έκφραση = ', expr)
end.
```

## Διαδικασίες (i)

- ◆ 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

## Διαδικασίες (ii)

- ◆ **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



## Διαδικασίες

(iii)

- ◆ **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

## Διαδικασίες

(iv)

- ◆ **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

## Διαδικασίες

(v)

```
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.
```

## Συναρτήσεις

(i)

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

## Συναρτήσεις

(ii)

### ◆ 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



## Εύρεση Μ.Κ.Δ.

(i)

### ◆ 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  $\min(a, b)$

## Εύρεση Μ.Κ.Δ.

(ii)

### ◆ Αλγόριθμος με αφαιρέσεις

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

## Εύρεση Μ.Κ.Δ.

(iii)

### ◆ 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  $\log(a+b)$

## Εύρεση Μ.Κ.Δ.

(iv)

```
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

## Εύρεση Μ.Κ.Δ.

(v)

### ◆ 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

## Εύρεση Μ.Κ.Δ.

(vi)

### ◆ Συγκεκριμενοποίηση (συνέχεια)

- ορθή είσοδος  
 $(a>0) \text{ and } (b>0)$
  - συμφωνημένη είσοδος για τέλος  
 $(a=0) \text{ and } (b=0)$
  - απομένουν μόνο: δηλώσεις, μηνύματα και σχόλια
- ### ◆ Συμπληρώνουμε τα παραπάνω στο πρόγραμμα

Δεύτερη  
προσέγγιση



## Εύρεση Μ.Κ.Δ.

(vii)

### ◆ Το τελικό πρόγραμμα

(\* Ευκλείδειος 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);
```

## Εύρεση Μ.Κ.Δ.

(viii)

### ◆ 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;  
    άθροιση αριθμών;  
    παρουσίαση αποτελεσμάτων;  
    μήνυμα για να ξέρει ο χρήστης  
      πώς θα συνεχίσει ή θα σταματήσει  
  until συμφωνημένη είσοδος για τέλος  
end.
```

Πρώτη  
προσέγγιση

## Αθροιστικό πρόγραμμα

(ii)

### ◆ Συγκεκριμενοποίηση

- άθροιση αριθμών

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

Δεύτερη  
προσέγγιση

## Αθροιστικό πρόγραμμα

(iii)

### ◆ Περαιτέρω συγκεκριμενοποίηση

- άθροιση αριθμών

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

Τρίτη  
προσέγγιση

## Αθροιστικό πρόγραμμα

(iv)

### ◆ 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');
```



## Αθροιστικό πρόγραμμα

(v)

### ◆ 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.
```

## Παρουσίαση και συντήρηση

(i)

### ◆ Ποιοτικά χαρακτηριστικά προγραμμάτων

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

## Παρουσίαση και συντήρηση

(ii)

### ◆ Στοίχιση

- Πρόγραμμα και υποπρογράμματα

|                                |                                  |                                 |
|--------------------------------|----------------------------------|---------------------------------|
| <b>program</b> ...<br>δηλώσεις | <b>procedure</b> ...<br>δηλώσεις | <b>function</b> ...<br>δηλώσεις |
| <b>begin</b><br>εντολές        | <b>begin</b><br>εντολές          | <b>begin</b><br>εντολές         |
| <b>end.</b>                    | <b>end</b>                       | <b>end</b>                      |

- Απλές εντολές

|                                     |                                      |                                    |
|-------------------------------------|--------------------------------------|------------------------------------|
| <b>if</b> ... <b>then</b><br>εντολή | <b>while</b> ... <b>do</b><br>εντολή | <b>for</b> ... <b>do</b><br>εντολή |
| <b>else</b><br>εντολή               |                                      |                                    |

## Παρουσίαση και συντήρηση

(iii)

### ◆ Στοίχιση (συνέχεια)

- Σύνθετες εντολές

|                                                                    |                                                                     |                                                                   |
|--------------------------------------------------------------------|---------------------------------------------------------------------|-------------------------------------------------------------------|
| <b>if</b> ... <b>then</b><br><b>begin</b><br>εντολές<br><b>end</b> | <b>while</b> ... <b>do</b><br><b>begin</b><br>εντολές<br><b>end</b> | <b>for</b> ... <b>do</b><br><b>begin</b><br>εντολές<br><b>end</b> |
| <b>else</b><br><b>begin</b><br>εντολές<br><b>end</b>               |                                                                     |                                                                   |

## Παρουσίαση και συντήρηση

(iv)

### ◆ Στοίχιση (συνέχεια)

- Σύνθετες εντολές (συνέχεια)

|                          |                                                                                                                                                                       |                                                                    |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| <b>repeat</b><br>εντολές | <b>case</b> ... <b>of</b><br>τιμή <sub>1</sub> : εντολή <sub>1</sub> ;<br>τιμή <sub>2</sub> : εντολή <sub>2</sub> ;<br>...<br>τιμή <sub>n</sub> : εντολή <sub>n</sub> | <b>with</b> ... <b>do</b><br><b>begin</b><br>εντολές<br><b>end</b> |
| <b>until</b> ...         | <b>end</b>                                                                                                                                                            |                                                                    |

## Είσοδος και έξοδος

### ◆ Συσκευές εισόδου/εξόδου

#### ◆ Είσοδος: εισαγωγή δεδομένων

- π.χ. από το πληκτρολόγιο

#### ◆ Έξοδος: παρουσίαση αποτελεσμάτων

- π.χ. στην οθόνη

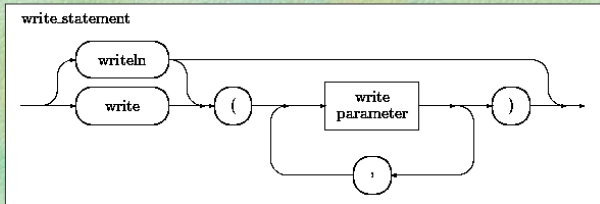
#### ◆ Προκαθορισμένες διαδικασίες της Pascal

- **read, readln**
- **write, writeln**



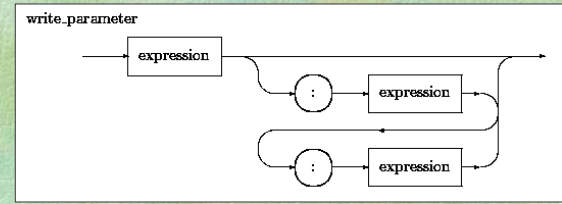
## Έξοδος (i)

### ◆ Έξοδος με μορφοποίηση



## Έξοδος (ii)

### ◆ Έξοδος με μορφοποίηση (συνέχεια)



## Έξοδος (iii)

### ◆ 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.
```

```
$
**$
****$
*****$
*****$
*****$
```

## Έξοδος (iv)

### ◆ 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.
```

3+ 6= 9



## Έξοδος (v)

### ◆ Ακέραιες τιμές, χωρίς μορφοποίηση

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

```
x = 42
y = -324
z = 123837
```

## Έξοδος (vi)

### ◆ Ακέραιες τιμές, με μορφοποίηση

```
writeln(42:4)
```

42

### ◆ 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



## Εξοδος

(vii)

### ◆ Πραγματικές τιμές, με μορφοποίηση

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

```
3.1416
```

### ◆ Παράδειγμα

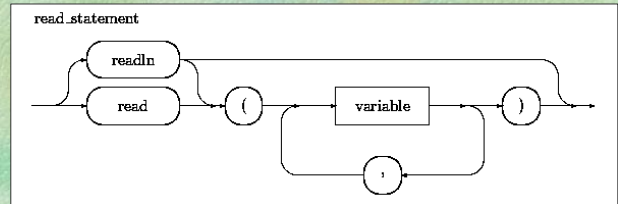
```
program temperatures(input,output);  
var i : integer; par, mos : real;  
begin par := 20.5; mos := -4.3;  
  writeln('Paris':10, par:6:1);  
  for i:=1 to 14 do write('-'); writeln;  
  writeln('Moscow':10, mos:6:1)  
end.
```

```
Paris 20.5  
-----  
Moscow -4.3
```

## Είσοδος

(i)

### ◆ Συντακτικό διάγραμμα



## Είσοδος

(ii)

### ◆ 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

## Είσοδος

(iii)

### ◆ 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.
```

## Είσοδος

(iv)

### ◆ 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 κειμένου

(i)

### ◆ Αρχεία εισόδου και εξόδου

### ◆ Παράδειγμα με αρχείο εισόδου

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



## Αρχεία κειμένου

(ii)

### ◆ Παράδειγμα με αρχείο εξόδου

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

## Ορθότητα

(i)

### ◆ Είδη ορθότητας

- Συντακτική
- Νοηματική
- Σημασιολογική

### ◆ Σημασιολογική ορθότητα ελέγχεται:

- με δοκιμές (testing)
- με μαθηματική επαλήθευση

## Ορθότητα

(ii)

### ◆ 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:

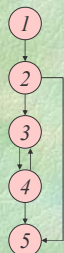
- Η συνάρτηση υπολογίζει το γινόμενο δυο φυσικών αριθμών x και y

## Ορθότητα

(iii)

### ◆ 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

## Ορθότητα

(iv)

### ◆ Βεβαιώσεις

```
(*1 - Βεβαίωση εισόδου: x ≥ 0, y ≥ 0 *)
z := 0;
(*2 : x ≥ 0, y ≥ 0, z = 0 *)
for i:=1 to x do
  (*3 - Αναλλοίωτη βρόχου:
  x ≥ 0, y ≥ 0, i ≤ x, z = y * (i-1) *)
  z := z+y
  (*4 : x ≥ 0, y ≥ 0, z = y * i *) ;
  (*5 - Βεβαίωση εξόδου: x ≥ 0, y ≥ 0, z = y * x *)
mult := z
```



### ◆ Επαλήθευση: για κάθε δυνατό τρόπο ροής 1 → 2, 2 → 3, 2 → 5, 3 → 4, 4 → 3, 4 → 5

## Ορθότητα

(v)

### ◆ 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
```



## Ορθότητα

(vi)

### ◆ Ροή ελέγχου

### ◆ Βεβαιώσεις

(\*1 - Βεβαίωση εισόδου:  $x$  : real,  $y$  : integer \*)

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

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

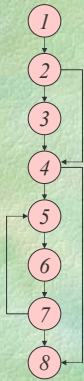
(\*4 :  $i \geq 0$ ,  $y^j = x^i$  \*)

(\*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:  $y^j = z$  \*)



## Ορθότητα

(vii)

### ◆ 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;
```

## Πίνακες

(i)

### ◆ 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**

## Πίνακες

(ii)

### ◆ 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'
```



## Πίνακες

(iii)

### ◆ 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;
```

## Πίνακες

(iv)

### ◆ 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]
```

## Γραμμική αναζήτηση

(i)

### ◆ Πρόβλημα (αναζήτησης): δίνεται ένας πίνακας ακεραίων **a** και ζητείται να βρεθεί αν υπάρχει ο ακεραίος **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.
```

## Γραμμική αναζήτηση

(ii)

### ◆ 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  $a \cdot n + b$  βήματα  $\Rightarrow$  γραμμική ( $a, b$  σταθερές,  $n$  το μέγεθος του πίνακα)

## Γραμμική αναζήτηση

(iii)

### ◆ 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')
```



## Γραμμική αναζήτηση

(iv)

### ◆ 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')
```

## Γραμμική αναζήτηση

(v)

### ◆ 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')
```

## Δυαδική αναζήτηση

(i)

- ◆ **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)

## Δυαδική αναζήτηση

(ii)

### ◆ 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 αύξουσα σειρά *)
  for i:=1 to howmany do read(a[i]);
  read(x);
  Αναζήτηση και εμφάνιση αποτελέσματος
end.
```

## Δυαδική αναζήτηση

(iii)

### ◆ 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')
```

## Ταξινόμηση

(i)

- ◆ **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
```



## Ταξινόμηση

(ii)

### ◆ 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  $n^2 \Rightarrow O(n^2)$

## Ταξινόμηση

(iii)

### ◆ 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
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
4 5 6 7 8 12 9
i = 5: 4 5 6 7 8 12 9
```

```
4 5 6 7 8 9 12
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] ...
```

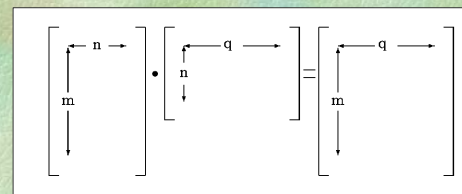
## Πολλαπλασιασμός πινάκων

(i)

### ◆ Δίνονται οι πίνακες: $a (m \times n)$ , $b (n \times q)$

### ◆ Ζητείται ο πίνακας: $c = a b (m \times q)$ όπου:

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



## Πολλαπλασιασμός πινάκων

(ii)

### ◆ 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
```

## Μαγικά τετράγωνα

(i)

### ◆ 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

|    |    |    |    |    |
|----|----|----|----|----|
| 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 |

### ◆ Πρόβλημα: κατασκευή μαγικού τετραγώνου ( $n \times n$ ) για περιττό $n$







## Εύρεση τετραγωνικής ρίζας (iii)

```
function sqrt(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 *) ;
  sqrt := 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  
abs(new-old) < eps
- Σχετική σύγκλιση  
abs(new-old) / new < eps

## Προκαθορισμένες συναρτήσεις

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

## Γραφικές παραστάσεις (i)

### ◆ Μορφοποίηση εξόδου (επανάληψη)

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

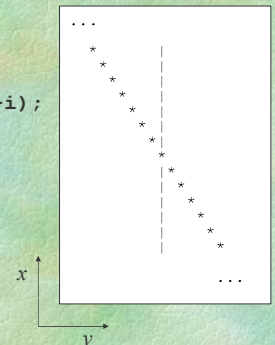
### ◆ Γραφικές παραστάσεις με χαρακτήρες

- 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)
```





## Γραφικές παραστάσεις

(iii)

◆ 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;

```

## Γραφικές παραστάσεις

(iv)

◆ Παράδειγμα (συνέχεια)

```

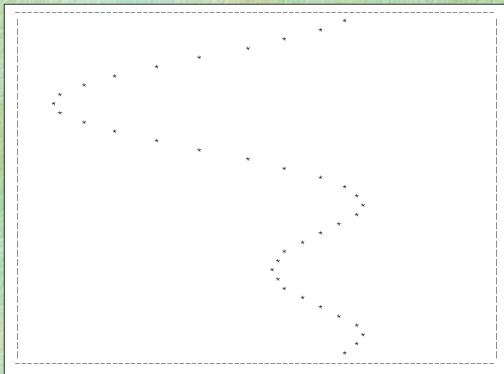
function f(j : integer) : integer;
  var x,y : real;
  begin x := pi * j / 18;
  y := 18 * sin(x) + 15 * cos(2*x);
  f := round(y)
  end;

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)



## Γραφικές παραστάσεις

(vi)

◆ 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;

```

## Γραφικές παραστάσεις

(vii)

◆ Παράδειγμα (συνέχεια)

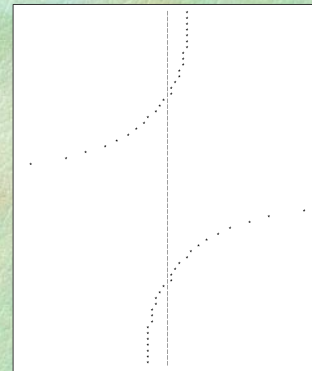
```

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)





## Γραφικές παραστάσεις

(ix)

### ◆ 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
```

## Γραφικές παραστάσεις

(x)

### ◆ 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
```

## Αναδρομή

(i)

### ◆ 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

$0! = 1$                        $(n+1)! = (n+1) * n!$

## Αναδρομή

(ii)

### ◆ 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



## Αναδρομή (iii)

### ◆ Αριθμοί Fibonacci

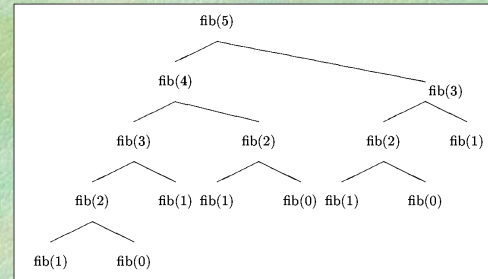
- $F_0 = 1$  ,  $F_1 = 1$
- $F_{n+2} = F_n + F_{n+1}$  ,  $\forall n \in \mathbb{N}$

### ◆ 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
end
```

## Αναδρομή (iv)

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



## Αναδρομή (v)

### ◆ 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
end
```

## Αναδρομή (vi)

### ◆ Συνάρτηση παρόμοια με του Ackermann

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

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

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

```
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
  end
  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
```

## Ταξινόμηση (i)

### ◆ 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

(ii)

- ◆ 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

(iii)

- ◆ 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
```

## Ταξινόμηση

(iv)

- ◆ 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);
```

## Ταξινόμηση

(v)

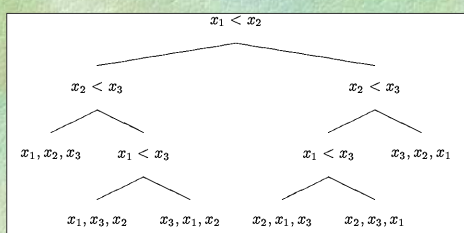
- ◆ Συγχώνευση (συνέχεια)

```
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
```

## Ταξινόμηση

(vi)

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



## Ταξινόμηση

(vii)

- ◆ Ταξινόμηση με διαμέριση (QuickSort)

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

- ◆ Διαμέριση (partition)

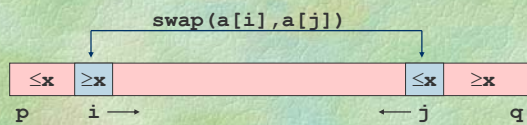
```
επιλογή ενός x := a[k];
repeat
  while a[i] < x do i := i+1;
  while x < a[j] do j := j-1;
  if i <= j then begin
    swap(a[i], a[j]); i := i+1; j := j-1
  end
until i > j
```



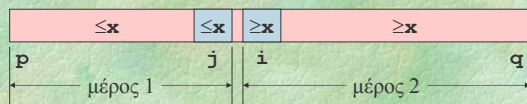
## Ταξινόμηση

(viii)

- ◆ Σε κάθε βήμα της διαμέρισης



- ◆ Μετά τη διαμέριση



## Τεχνολογία λογισμικού

- ◆ Software engineering

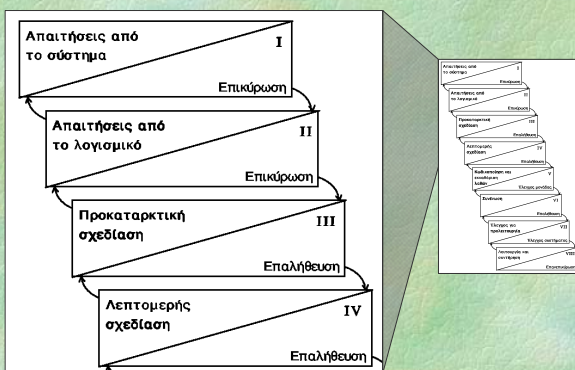
- ◆ Ανάπτυξη λογισμικού που να εξασφαλίζει:

- παράδοση μέσα σε προδιαγεγραμμένα χρονικά όρια
- κόστος μέσα σε προδιαγεγραμμένα όρια
- καλή ποιότητα
- αξιοπιστία
- δυνατή και όχι δαπανηρή συντήρηση

- ◆ Μοντέλα κύκλου ζωής λογισμικού

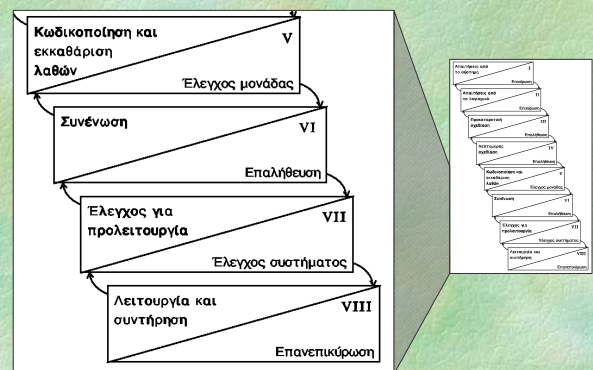
## Μοντέλο του καταρράκτη

(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

## Πέρασμα παραμέτρων, εμβέλεια

(ii)

- ◆ 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*

```

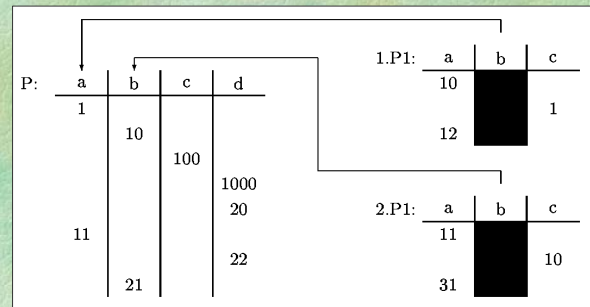
begin a:=1; b:=10; c:=100; d:=1000;
writeln(a:5,b:5,c:5,d:5);
p1(b,a); writeln(a:5,b:5,c:5,d:5);
p1(a,b); writeln(a:5,b:5,c:5,d:5)
end.

```

*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   |
| Output: | 11 | 10 | 100 | 20   |
|         | 31 | 21 | 10  | 22   |
|         | 11 | 21 | 100 | 22   |

### Πέρασμα παραμέτρων, εμβέλεια (vi)

#### ◆ 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)

#### ◆ Διάβαση και επεξεργασία όλων των χαρακτήρων που περιέχονται σε ένα αρχείο

```

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

```

#### ◆ Διάβαση 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

### Επεξεργασία κειμένου (ii)

#### ◆ 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.

```



## Επεξεργασία κειμένου (iii)

### ◆ 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)
```

## Επεξεργασία κειμένου (iv)

### ◆ 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)
```

## Επεξεργασία κειμένου (v)

### ◆ 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
```

## Επεξεργασία κειμένου (vi)

### ◆ 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
```

## Επεξεργασία κειμένου (vii)

### ◆ 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:

```
words of length 1    6  *****  
words of length 2    3  ***  
words of length 3    0  
...  
words of length 20   2  **
```

## Επεξεργασία κειμένου (viii)

### ◆ Παράδειγμα 3

```
program wordlength(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;
```



## Επεξεργασία κειμένου

(ix)

### ◆ 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;
  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.
```

## Επεξεργασία κειμένου

(xi)

### ◆ 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
```

## Επεξεργασία κειμένου

(xii)

### ◆ 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')
```

## Συμβολοσειρές

(i)

### ◆ 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)
```

## Συμβολοσειρές

(ii)

### ◆ 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   |



## Συμβολοσειρές

(iii)

### ◆ 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'

## Συμβολοσειρές

(iv)

### ◆ 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'

## Συμβολοσειρές

(v)

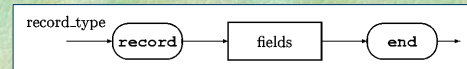
### ◆ Παράδειγμα

```
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.  
end.
```

## Εγγραφές

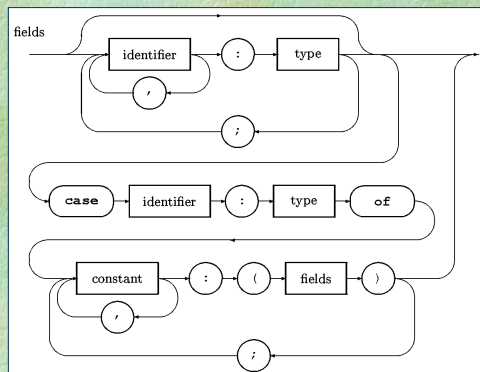
(i)

- ◆ 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)



## Εγγραφές

(iii)

### ◆ 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])
```



## Εγγραφές

(iv)

### ◆ 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;

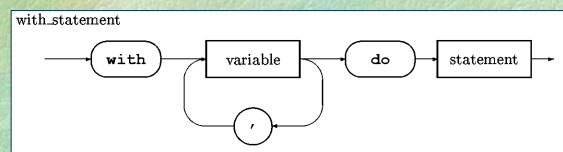
function cMult(x,y:complex) : complex;
begin
  cMult.re := x.re * y.re - x.im * y.im;
  cMult.im := x.re * y.im + x.im * y.re
end

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



## Εντολή with

(ii)

### ◆ 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
```



## Σύνολα

(i)

- ◆ 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];
```

## Σύνολα

(ii)

### ◆ Πράξεις με σύνολα

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

## Αρχεία

(i)

- ◆ 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
```

## Αρχεία

(ii)

### ◆ Αποθηκευτική μεταβλητή (buffer)

- $f^{\wedge}$  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)**

## Αρχεία

(iii)

### ◆ Λειτουργίες εισόδου και εξόδου

**get(f)    put(f)**

### ◆ Διάβασμα και γράψιμο

**read(f,x) ≡ begin x := f<sup>^</sup>; get(f) end**  
**write(f,x) ≡ begin f<sup>^</sup> := x; put(f) end**

### ◆ Έλεγχος τέλους αρχείου

**eof(f)**

## Αρχεία

(iv)

### ◆ 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.
```



## Αρχεία

(v)

### ◆ 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;
```

## Αρχεία

(vi)

### ◆ Παράδειγμα (συνέχεια)

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

## Αρχεία κειμένου

(i)

- ◆ 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)`

## Αρχεία κειμένου

(ii)

### ◆ 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
```

## Αρχεία κειμένου

(iii)

### ◆ Παράδειγμα (συνέχεια)

```
else  
  begin read(f, ch);  
        characters := characters + 1  
  end;  
close(f);  
writeln('lines: ', lines);  
writeln('characters: ', characters)  
end.
```

## Αρχεία κειμένου

(iv)

### ◆ 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
```



## Αρχεία κειμένου

(v)

### ◆ Παράδειγμα (συνέχεια)

```
readln (f) ;  
writeln (g)  
end ;  
close (f)  
end ;  
  
begin  
rewrite (g) ;  
copyOnce ;  
copyOnce ;  
close (g)  
end .
```

## Διαχείριση της μνήμης

(i)

### ◆ Στατικές μεταβλητές: γενικές ή τοπικές

- ο χώρος στη μνήμη όπου τοποθετούνται **δεσμεύεται** κάθε φορά που καλείται η ενότητα όπου δηλώνονται και **αποδεσμεύεται** στο τέλος της κλήσης

### ◆ Δυναμικές μεταβλητές

- ο χώρος στη μνήμη όπου τοποθετούνται **δεσμεύεται** και **αποδεσμεύεται** δυναμικά, δηλαδή με φροντίδα του προγραμματιστή
- η προσπέλαση σε δυναμικές μεταβλητές γίνεται με τη χρήση **δεικτών** (pointers)

## Διαχείριση της μνήμης

(ii)

### ◆ 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)

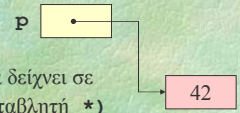
## Δείκτες

(i)

### ◆ 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)
```



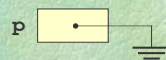
## Δείκτες

(ii)

### ◆ 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^)
```

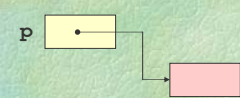
λάθος!

## Δυναμική παραχώρηση μνήμης

### ◆ Δέσμευση

- δημιουργία μιας νέας δυναμικής μεταβλητής

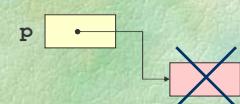
```
var p : ^integer;  
...  
new (p)
```



### ◆ Αποδέσμευση

- καταστροφή μιας δυναμικής μεταβλητής

```
...  
dispose (p)
```





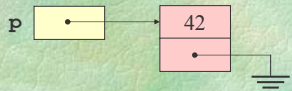
## Σύνθετες δυναμικές μεταβλητές (i)

### ◆ Παράδειγμα

```

type nodeptr = ^nodetype;
nodetype = record
    info : integer;
    next : nodeptr
end;

var p : nodeptr;
...
new(p);
p^.info := 42;
p^.next := nil
    
```



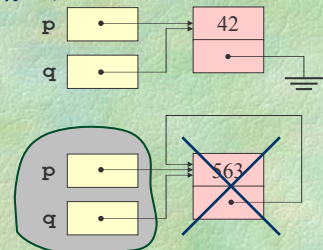
## Σύνθετες δυναμικές μεταβλητές (ii)

### ◆ Παράδειγμα (συνέχεια)

```

q := p;
q^.info := 563;
q^.next := q;

dispose(p)
    
```

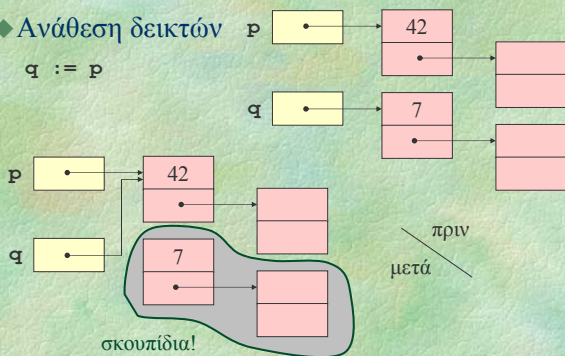


ξεκρέμαστοι δείκτες!

## Δείκτες και ανάθεση (i)

### ◆ Ανάθεση δεικτών

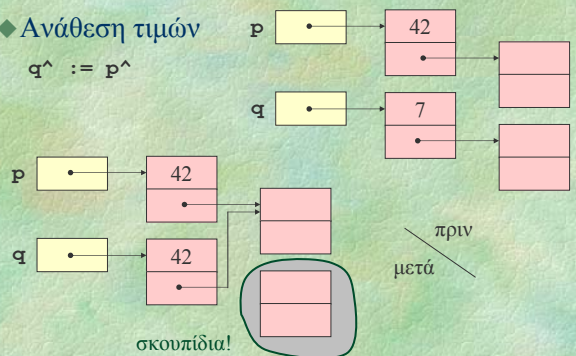
$q := p$



## Δείκτες και ανάθεση (ii)

### ◆ Ανάθεση τιμών

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



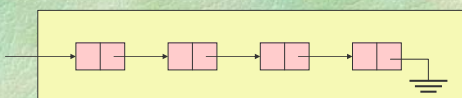
## Συνδεδεμένες λίστες (i)

### ◆ Είναι γραμμικές διατάξεις

### ◆ Κάθε κόμβος περιέχει:

- κάποια πληροφορία
- ένα σύνδεσμο στον επόμενο κόμβο

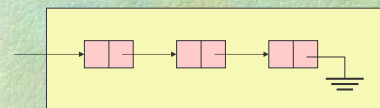
### ◆ Ο τελευταίος κόμβος έχει κενό σύνδεσμο



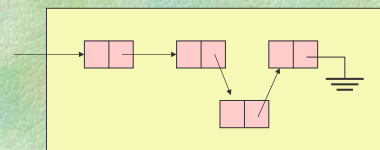
## Συνδεδεμένες λίστες (ii)

### ◆ Ευκολότερη προσθήκη στοιχείων

- πριν



- μετά



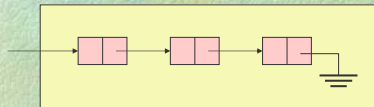


## Συνδεδεμένες λίστες

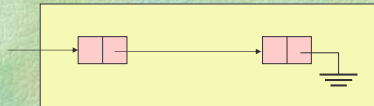
(iii)

### ◆ Ευκολότερη διαγραφή στοιχείων

- πριν



- μετά



## Συνδεδεμένες λίστες

(iv)

### ◆ Τύπος κόμβου συνδεδεμένης λίστας

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

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

```
var head : nodeptr
```

## Συνδεδεμένες λίστες

(v)

### ◆ Παράδειγμα κατασκευής λίστας

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

## Συνδεδεμένες λίστες

(v)

### ◆ 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.
```

## Συνδεδεμένες λίστες

(vi)

### ◆ Εκτύπωση λίστας

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

## Συνδεδεμένες λίστες

(vii)

### ◆ Εκτύπωση λίστας με αναδρομή

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



## Συνδεδεμένες λίστες (viii)

- ◆ Εκτύπωση λίστας αντίστροφα με αναδρομή

```

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

## Πολυπλοκότητα (i)

- ◆ Κόστος της εκτέλεσης ενός αλγορίθμου που επιλύει κάποιο πρόβλημα, συναρτήσει του μεγέθους του προβλήματος

- **χρόνος**: αριθμός υπολογιστικών βημάτων
- **χώρος**: απαιτούμενο μέγεθος μνήμης

- ◆ Συναρτήσεις πολυπλοκότητας

- θετικές και αύξουσες
- π.χ.  $f(n) = n(n-1)/2$

## Πολυπλοκότητα (ii)

- ◆ Άνω φράγμα:  $O$

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

- ◆ Κάτω φράγμα:  $\Omega$

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

- ◆ 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)

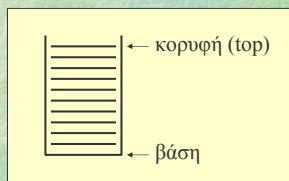
$$\begin{aligned}
 O(1) &< O(\log^* n) < O(\log n) < O(\sqrt{n}) \\
 &< O(n) < O(n \log n) \\
 &< O(n^2) < O(n^2 \log^5 n) \\
 &< O(n^3) < \dots < \text{Poly} \\
 &< O(2^n) < O(n!) < O(n^n) \\
 &< O(2^{2^n}) < \dots
 \end{aligned}$$

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

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

## Στοιβες (i)

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



## Στοιβες (ii)

- ◆ Αφηρημένος τύπος δεδομένων

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



## Στοιβες

(iii)

### ◆ 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
```

## Στοιβες

(iv)

### ◆ Προσθήκη στοιχείου

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

## Στοιβες

(v)

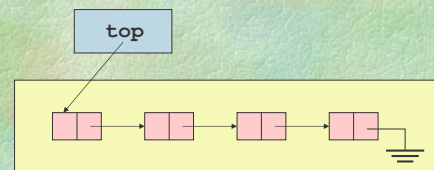
### ◆ Αφαίρεση στοιχείου

```
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
```

## Στοιβες

(vi)

### ◆ Υλοποίηση με απλά συνδεδεμένη λίστα



## Στοιβες

(vii)

### ◆ Υλοποίηση με απλά συνδεδεμένη λίστα

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

### ◆ Άδεια στοίβα

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

## Στοιβες

(viii)

### ◆ Προσθήκη στοιχείου

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



## Στοιβάς

(ix)

### ◆ Αφαίρεση στοιχείου

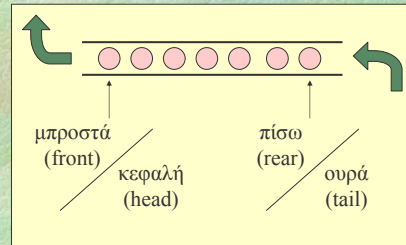
```
procedure pop (var s : stack;
               var data : integer;
               var nonempty : boolean);
var p : ^node;
begin
  if s = nil then
    nonempty := false
  else begin p := s;
            data := s^.info;
            s := s^.next;
            dispose (p);
            nonempty := true
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

## Ουρές

(i)

### ◆ First In First Out (FIFO)

ό,τι μπαίνει πρώτο, βγαίνει πρώτο



## Ουρές

(ii)

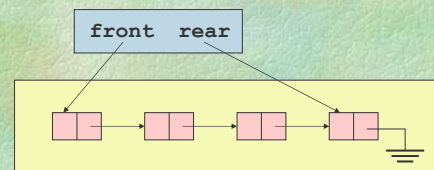
### ◆ Αφηρημένος τύπος δεδομένων

- 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

## Ουρές

(iii)

### ◆ Υλοποίηση με απλά συνδεδεμένη λίστα



## Ουρές

(iv)

### ◆ 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
```

## Ουρές

(v)

### ◆ 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
```



## Ουρές

(vi)

### ◆ 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
```

Σ. Ζάχος, Ν. Παπασπύρου

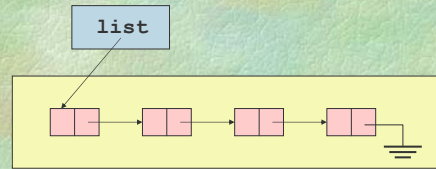
Προγραμματισμός Ηλεκτρονικών Υπολογιστών

289

## Γραμμικές λίστες

(i)

### ◆ Γενική μορφή απλά συνδεδεμένης λίστας



Σ. Ζάχος, Ν. Παπασπύρου

Προγραμματισμός Ηλεκτρονικών Υπολογιστών

290

## Γραμμικές λίστες

(ii)

### ◆ 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
```

Σ. Ζάχος, Ν. Παπασπύρου

Προγραμματισμός Ηλεκτρονικών Υπολογιστών

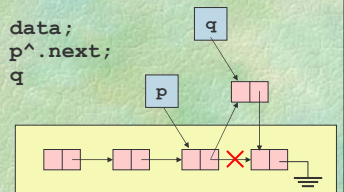
291

## Γραμμικές λίστες

(iii)

### ◆ 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
end
```



Σ. Ζάχος, Ν. Παπασπύρου

Προγραμματισμός Ηλεκτρονικών Υπολογιστών

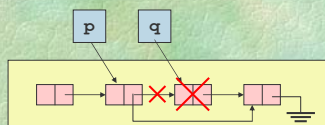
292

## Γραμμικές λίστες

(iv)

### ◆ 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
end
```



Σ. Ζάχος, Ν. Παπασπύρου

Προγραμματισμός Ηλεκτρονικών Υπολογιστών

293

## Γραμμικές λίστες

(v)

### ◆ 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 = data then
      found := true
    else
      p := p^.next;
    end
  search := p
end
```

Σ. Ζάχος, Ν. Παπασπύρου

Προγραμματισμός Ηλεκτρονικών Υπολογιστών

294



## Γραμμικές λίστες

(vi)

### ◆ 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  
  end
```

## Γραμμικές λίστες

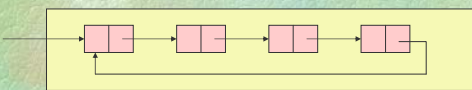
(vii)

### ◆ 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  
    end  
  end
```

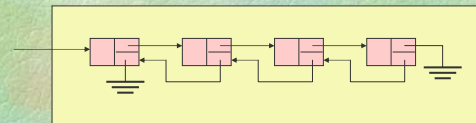
## Κυκλικές λίστες

### ◆ Ο επόμενος του τελευταίου κόμβου είναι πάλι ο πρώτος



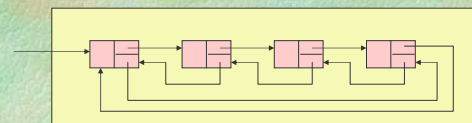
## Διπλά συνδεδεμένες λίστες

### ◆ 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



## Γράφοι

(i)

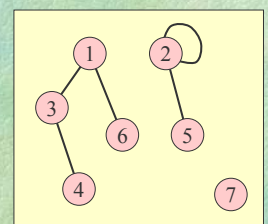
### ◆ Γράφος ή γράφημα (graph) $G = (V, E)$

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

### ◆ Παράδειγμα

$$V = \{1, 2, 3, 4, 5, 6, 7\}$$
$$E = \{(x, y) \mid x, y \in V, x+y=4 \text{ ή } x+y=7\}$$

### ◆ Γραφική παράσταση



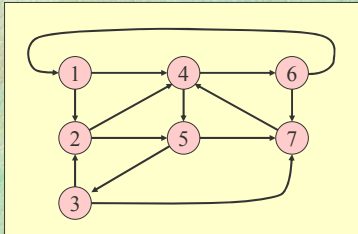


## Γράφοι

(ii)

### Κατευθυνόμενος γράφος (directed graph)

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

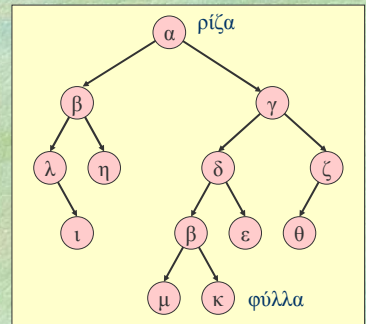


## Δυαδικά δέντρα

(i)

### Ειδικοί γράφοι της μορφής:

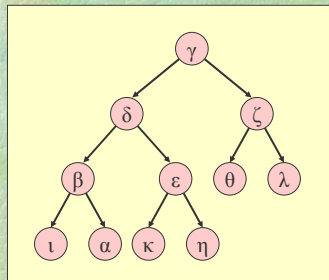
- ◆ Κάθε κόμβος έχει 0, 1 ή 2 παιδιά
- ◆ Ρίζα: ο αρχικός κόμβος του δένδρου
- ◆ Φύλλα: κόμβοι χωρίς παιδιά



## Δυαδικά δέντρα

(ii)

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

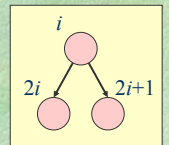


## Δυαδικά δέντρα

(iii)

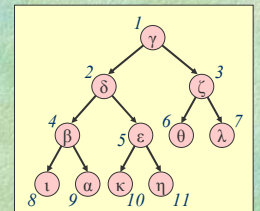
### Υλοποίηση με πίνακα

- 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'
```

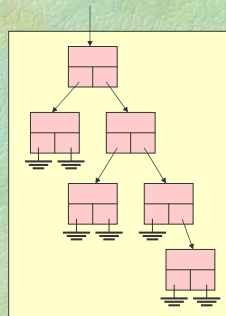
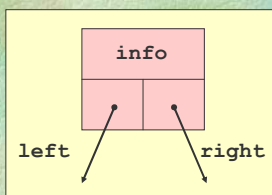


## Δυαδικά δέντρα

(iv)

### Υλοποίηση με δείκτες

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



## Δυαδικά δέντρα

(v)

### Διάσχιση όλων των κόμβων ενός δένδρου

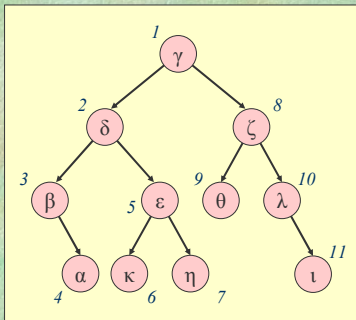
- προθεματική διάταξη (preorder)  
για κάθε υποδέντρο, πρώτα η ρίζα, μετά το αριστερό υποδέντρο και μετά το δεξιό
- επιθεματική διάταξη (postorder)  
για κάθε υποδέντρο, πρώτα το αριστερό υποδέντρο, μετά το δεξιό και μετά η ρίζα
- ενθεματική διάταξη (inorder)  
για κάθε υποδέντρο, πρώτα το αριστερό υποδέντρο, μετά η ρίζα και μετά το δεξιό



## Δυαδικά δέντρα

(vi)

### ◆ Διάσχιση preorder



Σ. Ζάχος, Ν. Παπασπύρου

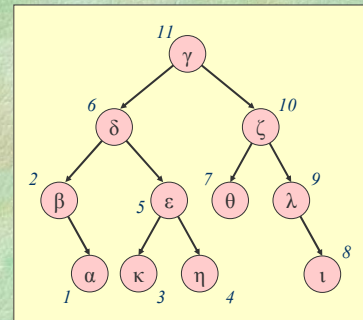
Προγραμματισμός Ηλεκτρονικών Υπολογιστών

307

## Δυαδικά δέντρα

(vii)

### ◆ Διάσχιση postorder



Σ. Ζάχος, Ν. Παπασπύρου

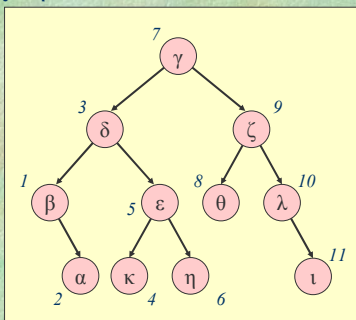
Προγραμματισμός Ηλεκτρονικών Υπολογιστών

308

## Δυαδικά δέντρα

(viii)

### ◆ Διάσχιση inorder



Σ. Ζάχος, Ν. Παπασπύρου

Προγραμματισμός Ηλεκτρονικών Υπολογιστών

309

## Δυαδικά δέντρα

(ix)

### ◆ Υλοποίηση 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)

Σ. Ζάχος, Ν. Παπασπύρου

Προγραμματισμός Ηλεκτρονικών Υπολογιστών

310

## 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

Σ. Ζάχος, Ν. Παπασπύρου

Προγραμματισμός Ηλεκτρονικών Υπολογιστών

311

## Το λειτουργικό σύστημα Unix

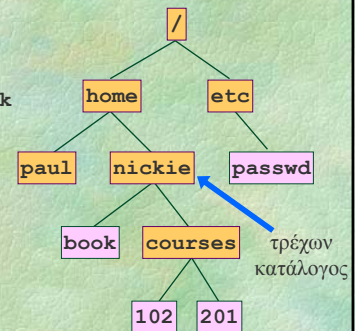
(ii)

### ◆ Απόλυτα ονόματα

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

### ◆ Σχετικά ονόματα

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



Σ. Ζάχος, Ν. Παπασπύρου

Προγραμματισμός Ηλεκτρονικών Υπολογιστών

312



## Το λειτουργικό σύστημα Unix (iii)

- ◆ Θετικά στοιχεία του Unix
  - ιεραρχικό σύστημα αρχείων
  - πολλοί χρήστες συγχρόνως (multi-user)
  - πολλές διεργασίες συγχρόνως (multi-tasking)
  - επικοινωνίες και υποστήριξη δικτύου
- ◆ Αρνητικά στοιχεία του Unix
  - κρυπτογραφικά ονόματα εντολών
  - περιορισμένη και συνθηματική βοήθεια

## Σύστημα αρχείων του Unix (i)

- ◆ Αντιγραφή αρχείων **cp**
  - `cp oldfile newfile`
  - `cp file1 file2 ... filen directory`
  - `cp -r directory1 directory2`
  - `cp -i oldfile newfile`
- ◆ Μετονομασία ή μετακίνηση αρχείων **mv**
  - `mv oldfile newfile`
  - `mv file1 file2 ... filen directory`
  - `mv -i oldfile newfile`

## Σύστημα αρχείων του Unix (ii)

- ◆ Διαγραφή αρχείων **rm**
  - `rm file1 file2 ... filen`
  - `rm -i file1 file2 ... filen`
  - `rm -f file1 file2 ... filen`
  - `rm -r directory`
- ◆ Δημιουργία directories **mkdir**
  - `mkdir directory1 ... directoryn`
- ◆ Διαγραφή άδειων directories **rmdir**
  - `rmdir directory1 ... directoryn`
- ◆ Αλλαγή directory **cd**
  - `cd directory`

## Σύστημα αρχείων του Unix (iii)

- ◆ Εμφάνιση πληροφοριών για αρχεία **ls**
  - `ls`
  - `ls file1 file2 directory3 ...`
  - 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 file1 file2 ... filen`
- ◆ Εμφάνιση περιεχομένων αρχείου ανά σελίδα **more less**
  - `more file1 file2 ... filen`
  - `less file1 file2 ... filen`

## Προγράμματα εφαρμογών Unix (ii)

- ◆ Εμφάνιση πρώτων γραμμών **head**
  - `head file1 file2 ... filen`
  - `head -10 file1 file2 ... filen`
- ◆ Εμφάνιση τελευταίων γραμμών **tail**
  - `tail file1 file2 ... filen`
  - `tail -10 file1 file2 ... filen`
- ◆ Πληροφορίες για το είδος αρχείου **file**
  - `file file1 file2 ... filen`
- ◆ Εμφάνιση ημερομηνίας και ώρας **date**
  - `date`



## Προγράμματα εφαρμογών Unix (iii)

- ◆ Εκτύπωση αρχείου **lpr**  
`lpr file1 file2 ... filen`
- ◆ Μεταγλωττιστής Pascal **pc**  
`pc -o executable program.p`  
`gpc -o executable program.p`
- ◆ Μεταγλωττιστής C **cc**  
`cc -o executable program.p`  
`gcc -o executable program.p`
- ◆ Επεξεργασία αρχείου κειμένου **vi**  
`vi file1 file2 ... filen`

## Βασική λειτουργία του vi (i)

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

## Βασική λειτουργία του vi (ii)

- ◆ Μετακίνηση μέσα σε αρχείο
  - `← ↓ ↑ →` κατά ένα χαρακτήρα
  - `h j k l` (ομοίως)
  - `w` μια λέξη δεξιά
  - `CTRL+F` μια σελίδα μετά
  - `CTRL+B` μια σελίδα πριν
  - `CTRL+D` μισή σελίδα μετά
  - `CTRL+U` μισή σελίδα πριν
  - `0 $` στην αρχή ή στο τέλος της γραμμής
  - `^` στον πρώτο χαρακτήρα της γραμμής

## Βασική λειτουργία του vi (iii)

- ◆ Μετακίνηση μέσα σε αρχείο (συνέχεια)
  - `- +` στην αρχή της προηγούμενης ή της επόμενης γραμμής
  - `( )` στην αρχή της προηγούμενης ή της επόμενης πρότασης
  - `{ }` στην αρχή της προηγούμενης ή της επόμενης παραγράφου
  - `n G` στην *n*-οστή γραμμή
  - `G` στην τελευταία γραμμή

## 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

## Βασική λειτουργία του vi (v)

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



## Βασική λειτουργία του vi (vi)

- ◆ Εύρεση συμβολοσειράς
  - / *xxx* εύρεση προς τα εμπρός
  - ? *xxx* εύρεση προς τα πίσω
  - n N** επόμενη εύρεση ορθής ή αντίθετης φοράς
- ◆ Άλλες εντολές
  - CTRL-L** επανασχεδίαση της εικόνας
  - u** ακύρωση της τελευταίας εντολής
  - .** επανάληψη της τελευταίας εντολής
  - J** συνένωση της τρέχουσας γραμμής με την επόμενη

## Βασική λειτουργία του vi (vii)

- ◆ Αντιγραφή και μετακίνηση κειμένου
  - yy** αντιγραφή μιας γραμμής στο buffer (**copy**)
  - n yy** αντιγραφή *n* γραμμών στο buffer
  - p P** επικόλληση των περιεχομένων του buffer κάτω ή πάνω από την τρέχουσα γραμμή (**paste**)
- ◆ Αποθήκευση και έξοδος
  - :w** αποθήκευση του αρχείου
  - :q** έξοδος
  - :wq** αποθήκευση του αρχείου και έξοδος
  - :q!** έξοδος χωρίς αποθήκευση

## Internet (i)

- ◆ Δίκτυο υπολογιστών (computer network)
- ◆ Ονόματα και διευθύνσεις υπολογιστών
  - Διεύθυνση IP 147.102.1.1
  - Όνομα **theseas.softlab.ece.ntua.gr**
    - ο υπολογιστής στο δίκτυο του Εργαστηρίου Τεχνολογίας Λογισμικού
    - στο δίκτυο της Σ.Η.Μ.Μ.Υ.
    - στο δίκτυο του Ε.Μ.Π.
    - στο δίκτυο της Ελλάδας
  - Επικράτειες (domains)

## Internet (ii)

- ◆ Ηλεκτρονικό ταχυδρομείο (e-mail)
  - ηλεκτρονική ταχυδρομική διεύθυνση **nickie@softlab.ntua.gr**
    - όνομα χρήστη
    - όνομα υπολογιστή ή επικράτειας
  - υπάρχει πληθώρα εφαρμογών που διαχειρίζονται το ηλεκτρονικό ταχυδρομείο

## Internet (iii)

- ◆ 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$
```

## Internet (iv)

- ◆ 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**



## Internet

(v)

### ◆ 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 διαβάσει τις ανακοινώσεις των άλλων και να βάλει την ανακοίνωσή του (posting)

## Internet

(vi)

### ◆ 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

## Internet

(vii)

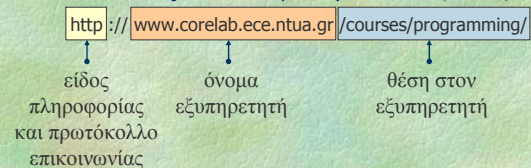
### ◆ Παγκόσμιος ιστός World-Wide Web (WWW)

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

## Internet

(viii)

### ◆ 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`