

## ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

<https://helios.ntua.gr/user/index.php?id=827>

Διδάσκοντες:	Στάθης Ζάχος	(zachos@cs.ntua.gr)
	Γεώργιος Γκούμας	(goumas@cslab.ece.ntua.gr)
	Δημήτρης Τσουμάκος	(dtsouma@cslab.ece.ntua.gr)
	Δημήτρης Φωτάκης	(fotakis@cs.ntua.gr)
	Μάριος Κόνιαρης	(mkoniari@dblab.ece.ntua.gr)
	Πέτρος Ποτίκας	(pprotik@cs.ntua.gr)
	Δώρα Σούλιου	(dsouliou@mail.ntua.gr)

### Διαφάνειες παρουσιάσεων

30/9/22

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

## 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)



Μαθηματικά

Σχεδίαση και  
κατασκευή

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 ⇒ στοιχειώδεις αριθμητικές πράξεις
- 1830–1840, **Babbage**, “αναλυτική μηχανή” ⇒ 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 ⇒ διαφορικές εξισώσεις
- ~1940, **Zuse**, ηλεκτρονική (ψηφιακή) υπολογιστική μηχανή ⇒ πρόγραμμα και δεδομένα, χωριστά
- 1945–1950, μοντέλο **von Neumann** ⇒ πρόγραμμα και δεδομένα, από κοινού
- 1950–σήμερα, ραγδαία ανάπτυξη της τεχνολογίας των **ηλεκτρονικών υπολογιστών**



## Εισαγωγή (xii)

### ◆ Κατασκευή υπολογιστών

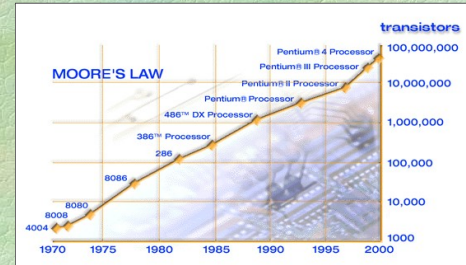
- 1952– main frames IBM 650, 7000, 360
- 1965– mini computers DEC PDP-8
- 1977– personal computers Apple II
- 1981 IBM PC
- 1983, 1984 Apple: Lisa, Macintosh
- 1985– internet
- 1990– world wide web
- 2000– PDA, smartphones, cloud, κ.λπ.

## 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*



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

## Εισαγωγή (xiv)

### ◆ Θεμέλια της πληροφορικής

- Μαθηματική λογική
- Αριστοτέλης: συλλογισμοί

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

- 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)

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

### ◆ 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

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

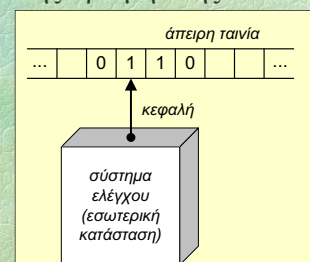
$$\begin{aligned} A \in A &\rightarrow A \notin A \\ A \notin A &\rightarrow A \in A \end{aligned}$$

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

## 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, υπολογισιμότητα





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

## Γλώσσες προγραμματισμού (iii)

### ◆ Pascal

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

### ◆ C

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

### • C++

- Bjarne Stroustrup (1983)
- Γενικού σκοπού, αντικειμενοστρεφής

pzhe1p



## Ασκήσεις

(i)

```
#include "pzhelp"
```

#1

```
PROGRAM {  
  WRITELN("hello world");  
}
```

```
#include "pzhelp"
```

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

```
#include "pzhelp"
```

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

```
#include "pzhelp"
```

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

## Ασκήσεις (Pascal)

(i)

```
program Hello1a(output);  
begin  
  writeln('hello world')  
end.
```

#1

```
program Hello1b(output);  
begin  
  writeln('hello ', 'world')  
end.
```

```
program Hello1c(output);  
begin  
  write('hello '); writeln('world')  
end.
```

```
program Hello1d(output);  
begin  
  write('hello world'); writeln  
end.
```

## Ασκήσεις (C)

(i)

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

#1

```
int main () {  
  printf("hello world\n");  
  return 0;  
}
```

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

```
int main () {  
  printf("hello " "world\n");  
  return 0;  
}
```

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

```
int main () {  
  printf("hello  
world\n");  
  return 0;  
}
```

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

```
int main () {  
  printf("hello world");  
  printf("\n");  
  return 0;  
}
```

## Ασκήσεις (C++)

(i)

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

#1

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

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

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

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

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

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

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

## Δυο λόγια για το #include "pzhelp"

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

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

Απλή C++

```
#include "pzhelp"
```

```
PROGRAM {  
  WRITELN("hello world");  
}
```

C++ & pzhelp

## Ασκήσεις

(ii)

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

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

#2

```
#include "pzhelp"
```

```
PROC hello() {  
  WRITELN("hello world");  
}
```

```
PROGRAM {  
  int i;  
  FOR(i, 1 TO 20) hello();  
}
```

#3

## Ασκήσεις (Pascal)

(ii)

```
program Hello2(output);  
  procedure hello;  
  begin  
    writeln('hello world')  
  end;  
begin  
  hello; hello  
end.
```

#3

#2

```
program Hello3(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("hello world\n");  
}
```

```
int main() {  
  hello();  
  return 0;  
}
```

#2

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

#3

## Ασκήσεις (C++)

(ii)

```
#include <iostream>  
using namespace std;  
void hello() {  
  cout << "hello world\n";  
}  
int main() {  
  hello();  
}
```

#2

#3

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

## Ασκήσεις

(iii)

```
#include "pzhhelp"  
const int n = 20;  
int i;  
PROC num_hello() {  
  WRITELN(i, "hello world");  
}  
PROGRAM {  
  FOR(i, 1 TO n) num_hello();  
}
```

#4

## Ασκήσεις (Pascal)

(iii)

```
program Hello4(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.
```

#4

## Ασκήσεις (C)

(iii)

```
#include <stdio.h>  
const int n = 20;  
int i;  
void num_hello() {  
  printf("%d hello world\n", i);  
}  
int main() {  
  for (i = 1; i <= n; i++)  
    num_hello();  
  return 0;  
}
```

#4



## Ασκήσεις (C++)

(iii)

```
#include <iostream>
using namespace std;
const int n = 20;
int i;

void num_hello() {
    cout << i << " hello world\n";
}

int main() {
    for (i = 1; i <= n; i++)
        num_hello();
}
```

#4

## Ασκήσεις

(iv)

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

PROGRAM {
    int n, i;
    WRITELN("Give number of greetings",
            "then press <enter>:");
    n = READ_INT();
    FOR(i, 1 TO n) hello();
}
```

#5

## Ασκήσεις (Pascal)

(iv)

```
program Hello5(input,output);
var i,n : integer;
procedure hello;
begin
    writeln('hello world')
end;
begin
    writeln('Give number of greetings ',
            'then press <enter>:');
    read(n);
    for i := 1 to n do hello
end.
```

#5

## Ασκήσεις (C)

(iv)

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

int main() {
    int n;
    printf("Give number of greetings "
            "then press <enter>:");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) hello();
    return 0;
}
```

#5

## Ασκήσεις (C++)

(iv)

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

void hello() {
    cout << "hello world\n";
}

int main() {
    int n;
    cout << "Give number of greetings "
            << "then press <enter>:";
    cin >> n;
    for (int i = 0; i < n; i++) hello();
}
```

#5

## Ασκήσεις

(v)

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

PROGRAM {
    int n, i;
    WRITELN("Give number of greetings",
            "then press <enter>:");
    n = READ_INT();
    if (n < 0)
        WRITELN("# is negative");
    else
        FOR(i, 1 TO n) hello();
}
```

#6



## Ασκήσεις (Pascal)

(v)

```
program Hello6(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.
```

#6

## Ασκήσεις (C)

(v)

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

#6

## Ασκήσεις (C++)

(v)

```
#include <iostream>
using namespace std;
void hello() {
  cout << "hello world\n";
}
int main() {
  int n;
  cout << "Give number of greetings "
    << "then press <enter>";
  cin >> n;
  if (n < 0)
    printf("#is negative\n");
  else
    for (int i = 0; i < n; i++) hello();
}
```

#6

## Δομή του προγράμματος

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

επικεφαλίδα

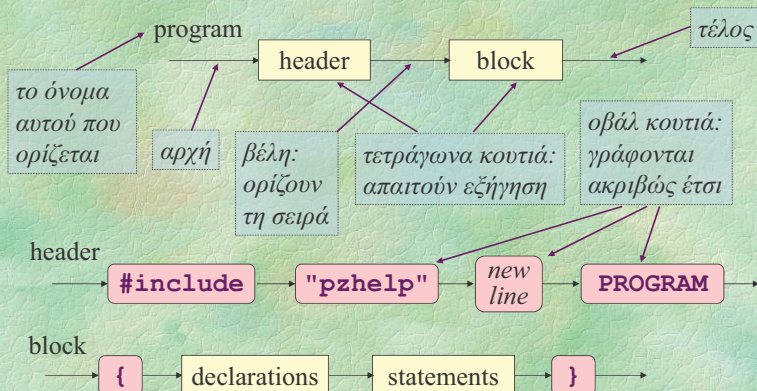
δηλώσεις

εντολές

σώμα = block

## Συντακτικά διαγράμματα

Περιγράφουν τη **σύνταξη** τμημάτων του προγράμματος



## Δηλώσεις μεταβλητών

(i)

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

```
int i;
```



## Δηλώσεις μεταβλητών (ii)

### ◆ Απλοί τύποι μεταβλητών

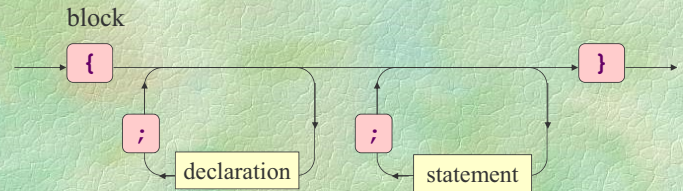
- **int** ακέραιοι αριθμοί 0, 1, -3
- **REAL** πραγματικοί αριθμοί 3.14
- **char** χαρακτήρες 'a'
- **bool** λογικές τιμές true, false

### ◆ Δήλωση περισσότερων μεταβλητών

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

## Συντακτικά διαγράμματα, ξανά

### ◆ Αναλυτικότερο συντακτικό διάγραμμα για το σώμα (block)



## Σχόλια

### ◆ Μίας ή περισσότερων γραμμών

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

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

## Τι σημαίνει ορθό πρόγραμμα (i)

### ◆ Συντακτική ορθότητα

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

### ◆ Συντακτικά σφάλματα στη C++

- εμφανίζονται όταν δεν ικανοποιούνται τα συντακτικά διαγράμματα
- παράδειγμα:  

```
{ PROGRAM } )WRITELN; ("hello world"
```

## Τι σημαίνει ορθό πρόγραμμα (ii)

### ◆ Νοηματική ορθότητα

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

### ◆ Νοηματικά σφάλματα στη C++

- εσφαλμένη χρήση τελεστών  

```
n = "a" * 3;
```
- χρήση μεταβλητών χωρίς δήλωση  

```
int n, i;  
n = i + j;
```

## Τι σημαίνει ορθό πρόγραμμα (iii)

### ◆ Σημασιολογική ορθότητα

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

### ◆ Σημασιολογικά σφάλματα στη C++

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

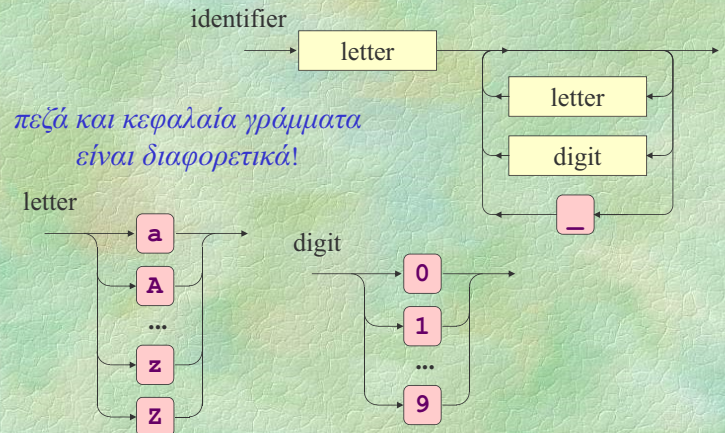
```
x1 = (-b + sqrt(b*b-4*a*c)) / (2*a);  
      sqrt  
      διαίρεση με το μηδέν
```



## Τι σημαίνει ορθό πρόγραμμα (iv)

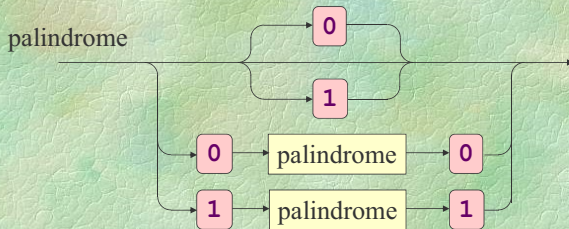
- ◆ Ο μεταγλωττιστής μπορεί να εντοπίσει σε ένα πρόγραμμα την ύπαρξη
  - συντακτικών σφαλμάτων
  - νοηματικών σφαλμάτων
- ◆ Τυπώνει κατάλληλα μηνύματα σφάλματος
- ◆ Ο προγραμματιστής είναι υπεύθυνος για
  - τη διόρθωση των παραπάνω
  - τον εντοπισμό και τη διόρθωση σημασιολογικών σφαλμάτων

## Συντακτικά διαγράμματα, ξανά



## Συντακτικά διαγράμματα, αναδρομικά

- ◆ Παλίνδρομο ή καρκινική συμβολοσειρά: διαβάζεται το ίδιο από αριστερά προς τα δεξιά και αντίστροφα: madam, 13431
- ◆ Παλίνδρομο με αλφάβητο {0, 1}



## Ανάθεση τιμής σε μεταβλητή

- ◆ Παραδείγματα αναθέσεων

```
n = 2;  
pi = 3.14159;  
done = true;  
ch = 'b';  
counter = counter + 1;  
x1 = (-b + sqrt(b*b-4*a*c)) / (2*a);
```

## Έξοδος στην οθόνη (i)

- ◆ Έξοδος στην οθόνη

```
WRITELN("Hello world!");  
WRITELN("Hello", "world!");
```
- ◆ Έξοδος χωρίς αλλαγή γραμμής

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

## Έξοδος στην οθόνη (ii)

- ◆ Έξοδος στην οθόνη

```
x = 6;  
3*x-1 = 17  
x*(x+1) = 42  
WRITE("x");  
WRITE(" = ");  
WRITE(x);  
WRITELN();  
WRITELN("3*x-1 =", 3*x-1);  
WRITELN("x*(x+1) =", x*(x+1));
```



## Έξοδος στην οθόνη (iii)

### ◆ Έξοδος στην οθόνη

```
WRITE (4) ;  
WRITELN (2) ;  
WRITELN (4, 2) ;  
WRITE (6) ;  
WRITE (6) ;  
WRITELN (6) ;  
WRITE (6, 6) ;  
WRITELN (6) ;
```

```
42  
4 2  
666  
6 66
```

## Είσοδος από το πληκτρολόγιο (i)

### ◆ Είσοδος από το πληκτρολόγιο

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

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

```
SKIP_LINE() ;
```

## Είσοδος από το πληκτρολόγιο (ii)

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

## Είσοδος από το πληκτρολόγιο (iii)

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

## Είσοδος από το πληκτρολόγιο (iv)

### ◆ Αποθηκευτικός χώρος (buffer)

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

## Είσοδος από το πληκτρολόγιο (v)

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



## Είσοδος από το πληκτρολόγιο (vi)

### ◆ Πρώτη εκτέλεση παραδείγματος

First: 3 ↵  
Second: 6 ↵  
Result: 9

### ◆ Δεύτερη εκτέλεση παραδείγματος

First: 3 6 ↵  
Second: Result: 9

## Είσοδος από το πληκτρολόγιο (vii)

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

## Είσοδος από το πληκτρολόγιο (viii)

### ◆ Πρώτη εκτέλεση παραδείγματος

First: 3 ↵  
Second: 6 ↵  
Result: 9

### ◆ Δεύτερη εκτέλεση παραδείγματος

First: 3 6 ↵  
Second: 6 ↵  
Result: 9

## Αριθμητικές παραστάσεις (i)

### ◆ Απλές παραστάσεις

- σταθερές και μεταβλητές

### ◆ Απλές πράξεις

- πρόσθεση, αφαίρεση +, -
- πολλαπλασιασμός \*
- διαίρεση πραγματικών αριθμών /
- ηλίκο ακέραιας διαίρεσης / *προσοχή!*
- υπόλοιπο ακέραιας διαίρεσης % **MOD**
- πρόσημα +, -

## Αριθμητικές παραστάσεις (ii)

### ◆ Παραδείγματα

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

### ◆ Πραγματική διαίρεση μεταξύ ακεραίων;

```
int x=42, y=17; WRITE( 1.0 * x / y );
```

## Αριθμητικές παραστάσεις (iii)

### ◆ Προτεραιότητα τελεστών

- π.χ.  $5+3*x-y \equiv 5+(3*x)-y$

### ◆ Προσεταιριστικότητα τελεστών

- π.χ.  $x-y+1 \equiv (x-y)+1$

### ◆ Σειρά εκτέλεσης των πράξεων

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



## 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	q	p OR q p    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

σύζευξη

p	NOT p ! p
false	true
true	false

διάζευξη

άρνηση

## Προτεραιότητα τελεστών (i)

Τελεστής	Σημασία	Προσεταιριστικότητα
+ - ! <b>NOT</b>	πρόσημα, λογική άρνηση	—
* / % <b>MOD</b>	πολλαπλασιασμός, διαίρεση	αριστερά
+ -	πρόσθεση, αφαίρεση	αριστερά
< <= >= >	σύγκριση	αριστερά
== !=	ισότητα	αριστερά
&& <b>AND</b>	λογική σύζευξη	αριστερά
<b>OR</b>	λογική διάζευξη	αριστερά

επάνω: μεγάλη προτεραιότητα

κάτω: μικρή προτεραιότητα

## Προτεραιότητα τελεστών (ii)

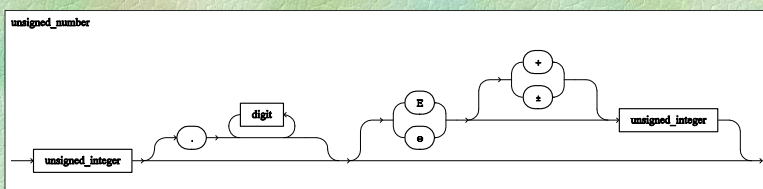
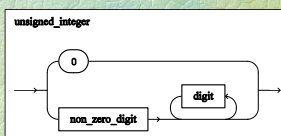
### ◆ Προτεραιότητα τελεστών

- π.χ. **NOT p AND q OR r**  
≡ ((**NOT p**) **AND q**) **OR r**
- π.χ. **x>3 AND NOT y+z==5** λάθος!  
≡ (**x>3**) **AND ((NOT y)+z==5)**
- π.χ. **x>3 AND NOT (y+z==5)** σωστό  
≡ (**x>3**) **AND (NOT ((y+z)==5))**

- Όταν δεν είμαστε σίγουροι, δε βλάπτει να χρησιμοποιούμε επιπλέον παρενθέσεις!

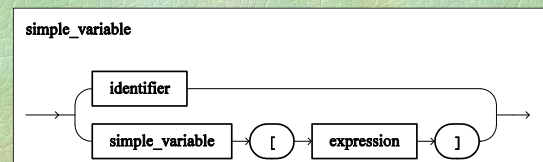
## Σύνταξη παραστάσεων (i)

### ◆ Σταθερές

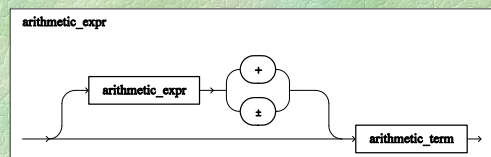


## Σύνταξη παραστάσεων (ii)

### ◆ Μεταβλητές (απλές)



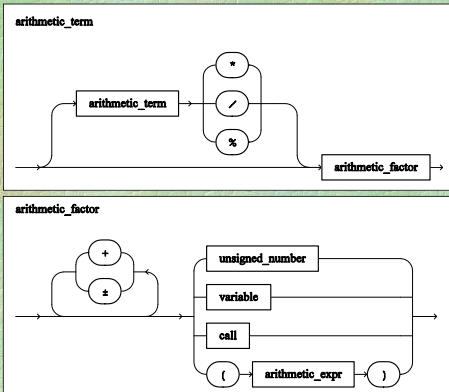
### ◆ Αριθμητικές παραστάσεις





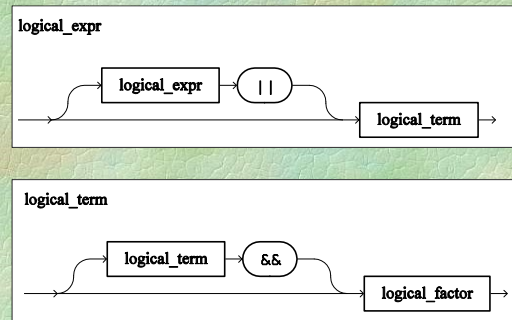
## Σύνταξη παραστάσεων (iii)

### ◆ Αριθμητικοί όροι και παράγοντες



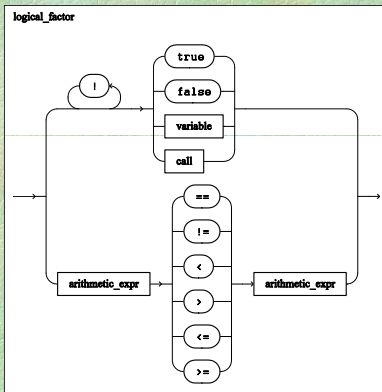
## Σύνταξη παραστάσεων (iv)

### ◆ Λογικές παραστάσεις και όροι



## Σύνταξη παραστάσεων (v)

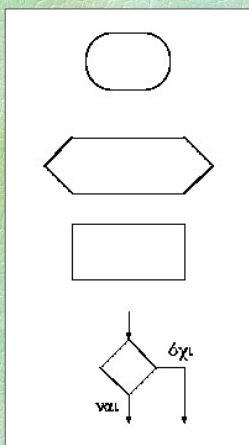
### ◆ Λογικοί παράγοντες



## Δομές ελέγχου

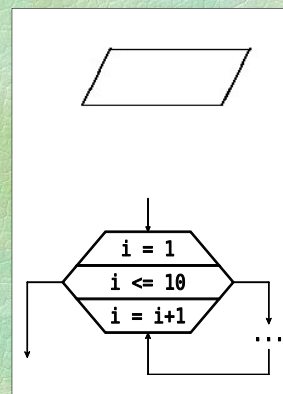
- ◆ Τροποποιούν τη σειρά εκτέλεσης των εντολών του προγράμματος
- ◆ Οι εντολές φυσιολογικά εκτελούνται κατά σειρά από την αρχή μέχρι το τέλος
- ◆ Με τις δομές ελέγχου επιτυγχάνεται:
  - ομαδοποίηση εντολών
  - εκτέλεση εντολών υπό συνθήκη
  - επανάληψη εντολών

## Λογικά διαγράμματα ροής (i)



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

## Λογικά διαγράμματα ροής (ii)

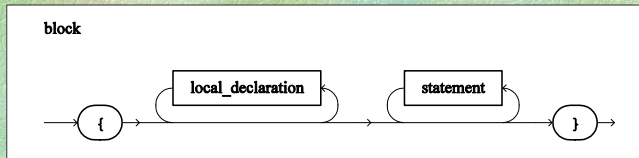


- ◆ Λειτουργία εισόδου/εξόδου
- ◆ Επανάληψη (βρόχος)



## Σύνθετη εντολή (i)

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



## Σύνθετη εντολή (ii)

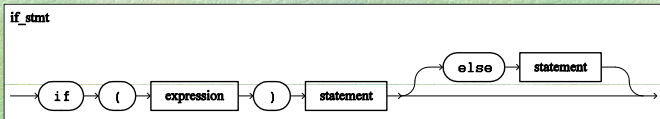
- ◆ Παραδείγματα

```
{
  int x=2, y=3, z=3;
  WRITELN(x, y, z);
}

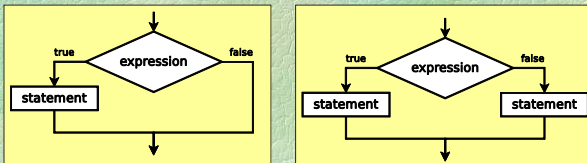
{
  a=2; b=3;
  {
    c=3;
    WRITE(a, b, c);
  }
  WRITELN();
}
```

## Εντολή if (i)

- ◆ Εκτέλεση εντολών υπό συνθήκη
- ◆ Συντακτικό διάγραμμα



- ◆ Λογικό διάγραμμα



## Εντολή if (ii)

- ◆ Παραδείγματα

```
if (amount >= x) amount = amount - x;

if (amount >= 1000000)
  WRITELN("Found a millionaire!");

if (year > 1900 AND year <= 2000)
  WRITE("20ός αιώνας");

if (x*x + y*y == z*z) {
  WRITELN("Pythagorean:", x, y, z);
  s = (z-x)*(z-y)/2;
  WRITELN("Perfect square:", s);
}
```

## Εντολή if (iii)

- ◆ Παραδείγματα

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

## Εντολή if (iv)

- ◆ Παραδείγματα (συνέχεια)

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

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

το παρακάτω είναι ισοδύναμο αλλά χειρότερο:

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



## Εντολή if (v)

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

### ◆ Παράδειγμα

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

## Εντολή switch (i)

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

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

## Εντολή switch (ii)

### ◆ Παραδείγματα

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

## Εντολή switch (iii)

### ◆ Περισσότερα παραδείγματα

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

## Εντολή switch (iv)

### ◆ Οι μέρες μέχρι την πρωτοχρονιά

```
r = 0;
switch (month) {
  case 1: r = r + 31; // συνεχίζει...
  case 2: r = r + 28; // συνεχίζει...
  case 3: r = r + 31; // συνεχίζει...
  case 4: r = r + 30; // συνεχίζει...
  case 5: r = r + 31; // συνεχίζει...
  ...
  case 11: r = r + 30; // συνεχίζει...
  case 12: r = r + 31; // συνεχίζει...
}
r = r - day + 1;
WRITELN("Μένουν", r, "μέρες!");
```

Χωρίς **break**

## Εντολή for (i)

### ◆ Μαθαίνω να μετρώ

```
PROGRAM { // counting
  int i;
  WRITELN("Look:");
  FOR (i, 1 TO 10)
    WRITELN(i);
}
```

- 1 Για όλες τις τιμές της μεταβλητής **i**
- 2 Από το 1 μέχρι και το 10
- 3 Εκτέλεσε το σώμα
- 4 Αύξησε κατά 1 την τιμή της **i**
- 5 Επανάλαβε!

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



## Εντολή for (ii)

- ◆ Το **FOR** υπάρχει μόνο στο **pzheIp**
  - ◆ Στη C++ υπάρχει η εντολή **for**
- FOR**(i, 1 TO 10) ...  
είναι ισοδύναμο με  
**for** (i = 1; i <= 10; i++) ...

## Εντολή for (iii)

- ◆ Μαθαίνω να μετρώ (ξανά)

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

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

- 1 Ξεκίνησε τη μεταβλητή **i** από την τιμή 1
- 2 Όσο η τιμή της **i** δεν ξεπερνά το 10
- 3 Εκτέλεσε το σώμα
- 4 Αύξησε κατά 1 την τιμή της **i**
- 5 Επανάλαβε!

## Εντολή for (iv)

- ◆ Μαθαίνω να μετρώ (ξανά)

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

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

## Εντολή for (v)

- ◆ Δυνάμεις του δύο

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

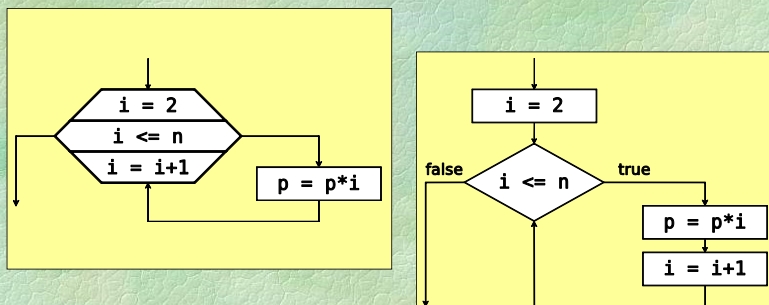
2<sup>0</sup> = 1  
2<sup>1</sup> = 2  
2<sup>2</sup> = 4  
2<sup>3</sup> = 8  
2<sup>4</sup> = 16  
2<sup>5</sup> = 32  
2<sup>6</sup> = 64  
2<sup>7</sup> = 128  
2<sup>8</sup> = 256  
2<sup>9</sup> = 512  
2<sup>10</sup> = 1024

Αναλλοίωτη:  $p = 2^i$

## Εντολή for (vi)

- ◆ Παραγοντικό

$$n! = 1 \times 2 \times 3 \times \dots \times n \quad 0! = 1$$



## Εντολή for (vii)

- ◆ Παραγοντικό

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

Give n: 1.↵  
1 ! = 1  
Give n: 4.↵  
4 ! = 24  
Give n: 7.↵  
7 ! = 5040  
Give n: 12.↵  
12 ! = 479001600

Give n: 17.↵  
17 ! = -288522240

*Υπερχείλιση!*

Αναλλοίωτη:  $p = i!$



## Εντολή for

(viii)

### ◆ Βλέπω αστεράκια

```
FOR(i, 1 TO 5) {
  FOR(j, 1 TO 10) WRITE("*");
  WRITELN();
}
```

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

```
FOR(i, 1 TO 5) {
  FOR(j, 1 TO 2*i) WRITE("*");
  WRITELN();
}
```

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

## Εντολή for

(ix)

### ◆ Ντόμινο

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



## Εντολή for

(x)

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

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

## Εντολή for

(xi)

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

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

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

## Εντολή for

(xii)

- ◆ Λίγα μαθηματικά δε βλάπτουν...

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

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

```
Give n: 6.↓
Total 28 pieces.
Give n: 17.↓
Total 171 pieces.
Give n: 42.↓
Total 946 pieces.
```

## Εντολή for

(xiii)

- ◆ Υπολογίζουμε το ίδιο με 3 διαφορετικούς τρόπους

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

domino2    dominol    domino0

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



## Εντολή for

(xiv)

### ◆ Παραλλαγές: αύξηση και μείωση

- ◆ **FOR(i, lower TO upper)**  
`for (i = lower; i <= upper; i++)`  
αν  $lower \leq upper$ , θα γίνουν  $upper - lower + 1$   
επαναλήψεις, αλλιώς καμία
- ◆ **FOR(i, upper DOWNTO lower)**  
`for (i = upper; i >= lower; i--)`  
αν  $lower \leq upper$ , θα γίνουν  $upper - lower + 1$   
επαναλήψεις, αλλιώς καμία

## Εντολή for

(xv)

### ◆ Παραλλαγές: αύξηση και μείωση με βήμα

- ◆ **FOR(i, 1 TO 20 STEP 3)**  
`for (i = 1; i <= 20; i += 3)`  
i παίρνει τις τιμές: 1, 4, 7, 10, 13, 16, 19
- ◆ **FOR(i, 100 DOWNTO 50 STEP 5)**  
`for (i = 100; i >= 50; i -= 5)`  
i παίρνει τις τιμές: 100, 95, 90, 85, ..., 60, 55, 50

## Εντολή for

(xvi)

### ◆ Ειδικές περιπτώσεις: μία φορά

```
FOR(i, 10 TO 10)
for (i = 10; i <= 10; i++)
FOR(i, 10 DOWNTO 10)
for (i = 10; i >= 10; i--)
```

### ◆ Ειδικές περιπτώσεις: καμία φορά

```
FOR(i, 12 TO 10)
for (i = 12; i <= 10; i++)
FOR(i, 10 DOWNTO 12)
for (i = 10; i >= 12; i--)
```

## Εντολή for

(xvii)

### ◆ Κακή ιδέα #1:

- ◆ Να αλλάζουμε τη μεταβλητή ελέγχου (π.χ. με ανάθεση) μέσα στο σώμα του βρόχου

```
FOR(i, 1 TO 10) {
  if (i > 5) i = 20;
  WRITELN(i);
}
```

- ◆ Δεν απαγορεύεται, κάνει όμως δύσκολη την κατανόηση των βρόχων!

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

## Εντολή for

(xviii)

### ◆ Κακή ιδέα #2:

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

```
n = 10;
FOR(i, 1 TO n) {
  n--; // ή χειρότερα n++;
  WRITELN(i);
}
```

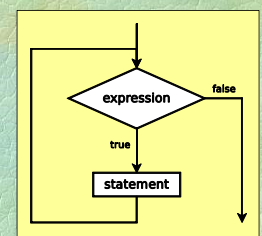
- ◆ Ούτε αυτό απαγορεύεται, κάνει όμως δύσκολη την κατανόηση των βρόχων!

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

## Εντολή while

(i)

- ◆ Βρόχος όσο ικανοποιείται μια συνθήκη
- ◆ Λογικό διάγραμμα



- ◆ Συντακτικό διάγραμμα





## Εντολή while (ii)

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

## Εντολή while (iii)

- ◆ Δυνάμεις του δύο, ξανά

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

```
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
...
2^22 = 4194304
2^23 = 8388608
```

Αναλλοίωτη:  $p = 2^i$

## Εντολή while (iv)

- ◆ Άπειρος βρόχος

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

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

Break

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

## Εντολή while (v)

- ◆ Άπειρος βρόχος, λιγότερο προφανής

```
PROGRAM { // another_infinite_loop
    int x = 17;
    while (x > 0)
        x = (x + 42) MOD 2012;
}
```

x
17
59
101
143
185
...
1991
21
63
105
...

Αναλλοίωτη: το x είναι θετικός και περιττός ακέραιος

## Εντολή while (vi)

- ◆ Πρώτοι αριθμοί

```
PROGRAM { // primes
    int p, t;
    WRITELN(2);
    FOR(p, 3 TO 1000 STEP 2) {
        t = 3;
        while (p MOD t != 0)
            t = t + 2;
        if (p == t) WRITELN(p);
    }
}
```

Output	p	t
2	3	3
3	5	3
5	5	5
7	7	3
11	5	7
...	7	7
997	9	3
...	11	3
...	5	7
...	7	7
...	...	...
997	997	997
999	999	3

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

## Εντολή while (vii)

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

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

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

Πολυπλοκότητα:  $O(\min(a, b))$



## Εντολή while (viii)

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

## Εντολή while (ix)

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

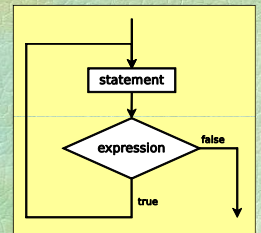
## Εντολή while (x)

```
PROGRAM { // gcd
  int a, b;
  WRITE("Give a: "); a = READ_INT();
  WRITE("Give b: "); b = READ_INT();
  WRITE("gcd(", a, ", ", b, ") =");
  a = abs(a); b = abs(b);
  while (a > 0 AND b > 0)
    if (a > b) a = a % b; else b = b % a;
  WRITELN(a+b);
}
```

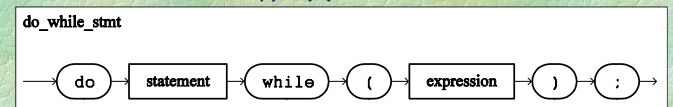
## Εντολή do ... while (i)

- ◆ Βρόχος με τη συνθήκη να αποτιμάται στο τέλος κάθε επανάληψης

- ◆ Λογικό διάγραμμα



- ◆ Συντακτικό διάγραμμα



## Εντολή do ... while (ii)

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

## Εντολή do ... while (iii)

- ◆ Αριθμοί Fibonacci

$$F_0 = 0, F_1 = 1$$

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

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

- ◆ Πρόβλημα: ποιος είναι ο μεγαλύτερος αριθμός Fibonacci που δεν υπερβαίνει το  $n$ ;

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

- ◆ NB: Η ακολουθία Fibonacci είναι **αύξουσα**



## Εντολή do ... while (iv)

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

```
Give n: 20.␣
13
Give n: 100.␣
89
Give n: 987.␣
987
```

*Αναλλοίωτη;*

## Εντολή do ... while (v)

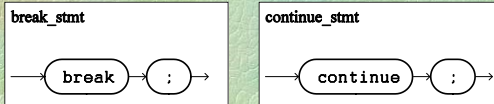
```
PROGRAM { // bigsum
  int sum, number; char symbol;
  do {
    sum = 0;
    do {
      number = READ_INT();
      sum = sum + number;
      do symbol = getchar();
      while (symbol != '+' AND
             symbol != '=');
    } while (symbol == '+');
    WRITELN(sum);
  } while (true);
}
```

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

Break

## Εντολές break και continue (i)

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



## Εντολές break και continue (ii)

- ◆ Η ατυχής εικασία...  
Ένας φίλος μας μαθηματικός ισχυρίζεται ότι για κάθε πρώτο αριθμό  $p$  ισχύει:  
 $(17p) \bmod 4217 \neq 42$
- ◆ Θα προσπαθήσουμε να βρούμε **αντιπαράδειγμα!**
- ◆ Δηλαδή έναν πρώτο αριθμό  $p$  τέτοιον ώστε  
 $(17p) \bmod 4217 = 42$

## Εντολές break και continue (iii)

- ◆ Θα τροποποιήσουμε το πρόγραμμα υπολογισμού των πρώτων αριθμών

```
PROGRAM { // primes
  int p, t;
  WRITELN(2);
  FOR(p, 3 TO 1000 STEP 2) {
    t = 3;
    while (p MOD t != 0) t = t + 2;
    if (p == t) WRITELN(p);
  }
}
```

## Εντολές break και continue (iv)

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

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

μόλις βρεις αντιπαράδειγμα σταμάτησε

Counterexample: 95009

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



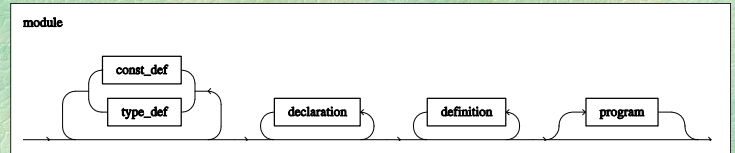
## Κενή εντολή

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

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

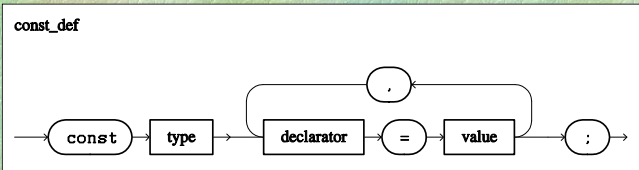
## Δομή του προγράμματος, ξανά

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



## Σταθερές (i)

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



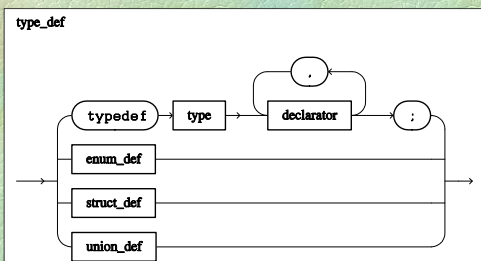
## Σταθερές (ii)

- ◆ Παραδείγματα

```
const int N = 100000;  
const REAL pi = 3.1415926535,  
             e = 2.7182818284;  
const char SPACE = ' ';
```
- ◆ Χρήση αντί των σταθερών εκφράσεων
  - ◆ π.χ. **FOR(i, 1 TO N) ...**
- ◆ Προκαθορισμένες σταθερές
  - ◆ π.χ. **INT\_MIN, INT\_MAX**

## Συνώνυμα τύπων (i)

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



## Συνώνυμα τύπων (ii)

- ◆ Παραδείγματα

```
typedef int number;  
typedef bool bit;  
typedef REAL real;
```
- ◆ Χρήση αντί των τύπων

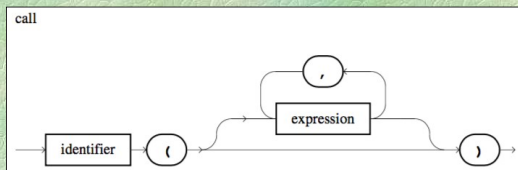
```
number n;  
bit b;   real r;
```
- ◆ Προκαθορισμένοι τύποι
  - ◆ π.χ. **int, REAL, bool, char**



## Διαδικασίες

(i)

- ◆ Ορίζονται στο τμήμα δηλώσεων
- ◆ Κάθε ορισμός διαδικασίας περιέχει:
  - την επικεφαλίδα της
  - 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 έξω από υποπρογράμματα και έχουν εμβέλεια σε ολόκληρο το module

## Διαδικασίες

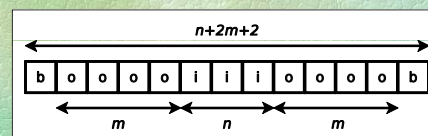
(iii)

- ◆ **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

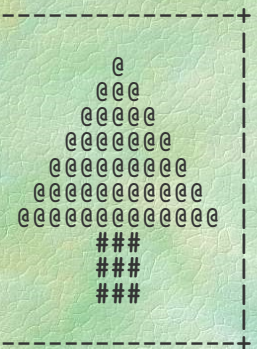
## Διαδικασίες

(iv)

- ◆ Χριστουγεννιάτικη καρτ ποστάλ
- ◆ Κάθε γραμμή έχει τη μορφή:



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



## Διαδικασίες

(v)

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

```
{  
  int i;  
  WRITE(border); // αριστερό πλαίσιο  
  FOR(i, 1 TO m) WRITE(outside);  
  FOR(i, 1 TO n) WRITE(inside);  
  FOR(i, 1 TO m) WRITE(outside);  
  WRITELN(border); // δεξιό πλαίσιο  
}
```

όνομα διαδικασίας

εμβέλεια του i

τυπικές παράμετροι

## Διαδικασίες

(vi)

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

```
PROGRAM { // tree_postcard  
  int i;  
  line('+', 15, '-', 0, ' '); // πάνω πλαίσιο  
  line('|', 15, '|', 0, ' ');  
  FOR(i, 1 TO 13 STEP 2)  
    line('|', i, '@', (15-i)/2, ' ');  
  FOR(i, 1 TO 3)  
    line('|', 3, '#', 6, ' ');  
  line('|', 15, '|', 0, ' ');  
  line('+', 15, '-', 0, ' '); // κάτω πλαίσιο  
}
```

πραγματικές παράμετροι



## Διαδικασίες

(vii)

### ◆ Σύγκρουση ονομάτων

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

### ◆ Εκτέλεση με το χέρι

### ◆ Trace tables

## Διαδικασίες

(viii)

```
int a, b, c;
PROC p42 (int y, int b) {
    int c = 42; WRITELN(a, b, c, y);
    a = a + b; c = c + 1; b = c + b; y = y-1;
    WRITELN(a, b, c, y);
}
PROC p17 (int a, int x) {
    int b = 17; WRITELN(a, b, c, x);
    p42(b, x); WRITELN(a, b, c, x);
}
PROGRAM { // proc_example
    a = 1; b = 2; c = 3; p17(b, c); p42(c, a);
}
```

## Διαδικασίες

(ix)

Global	a	b	c
	1	2	3
	4		
	8		

p17	a	x	b
	2	3	17

p42	y	b	c
	17	3	42
	16	46	43

p42	y	b	c
	3	4	42
	2	47	43

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

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

Output

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

Global	a	b	c
	1	2	3
	4		
	8		

p17	a	x	b
	2	3	17

p42	y	b	c
	17	3	42
	16	46	43

p42	y	b	c
	3	4	42
	2	47	43

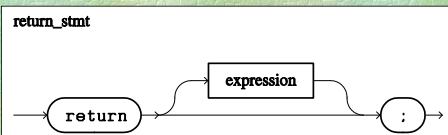
```
PROC p17 (int a, int x) {
    int b = 17;
    WRITELN(a, b, c, x);
    p42(b, x);
    WRITELN(a, b, c, x);
}
```

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

## Συναρτήσεις

(i)

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



## Συναρτήσεις

(ii)

```
FUNC int gcd (int a, int b) {
    a = abs(a); b = abs(b);
    while (a > 0 AND b > 0)
        if (a > b) a = a % b;
        else b = b % a;
    return a+b;
}
PROGRAM { // gcd_func
    int x, y;
    WRITE("Give x: "); x = READ_INT();
    WRITE("Give y: "); y = READ_INT();
    WRITELN("gcd is:", gcd(x, y));
}
```



## Δομημένος προγραμματισμός

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

## 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

## Παρουσίαση και συντήρηση (i)

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

## Παρουσίαση και συντήρηση (ii)

- ◆ **Στοίχιση**
  - Πρόγραμμα και υποπρογράμματα

```
PROGRAM {      PROC ... {      FUNC ... {
  δηλώσεις      δηλώσεις      δηλώσεις
  εντολές        εντολές        εντολές
}                }                }
```
  - Απλές εντολές

```
if (...) εντολή      while (...) εντολή
else εντολή          FOR (...) εντολή
                     for (...) εντολή
```

## Παρουσίαση και συντήρηση (iii)

- ◆ **Στοίχιση (συνέχεια)**
  - Σύνθετες εντολές

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

## Παρουσίαση και συντήρηση (iv)

- ◆ **Στοίχιση (συνέχεια)**
  - Σύνθετες εντολές (συνέχεια)

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



## Έξοδος με μορφοποίηση

### ◆ Ακέραιες τιμές

```
WRITELN(FORM(42, 4));
```

```
□ □ □ 4 2
```

### ◆ ... αλλά και οτιδήποτε άλλο

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

```
□ □ □ h e l l o
```

### ◆ Πραγματικές τιμές

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

```
□ □ 3 . 1 4 1 6
```

## Αρχεία κειμένου

### ◆ Ανακατεύθυνση εισόδου και εξόδου

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

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

```
  n = READ_INT();
  FOR(i, 1 TO n)
    sum = sum + READ_INT();
  WRITELN(sum);
}
```

## Τακτικοί τύποι

### ◆ Οι τύποι **int**, **bool** και **char**

### ◆ Απαριθμητοί τύποι

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

```
color c = green;
```

```
day d = fri;
```

### ◆ Πράξεις με τακτικούς τύπους

- τελεστές σύγκρισης ==, !=, <, >, <=, >=

## Πίνακες

(i)

### ◆ Δομημένη μεταβλητή: αποθηκεύει μια συλλογή από τιμές δεδομένων

### ◆ Πίνακας (array): δομημένη μεταβλητή που αποθηκεύει πολλές τιμές του ίδιου τύπου

```
int n[5];
```

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

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

και έχουν τύπο **int**

## Πίνακες

(ii)

### ◆ Παραδείγματα

```
REAL a[10];
int b[20];
char c[30];
```

```
...
```

```
a[1] = 4.2;
```

```
a[3] = READ_REAL();
```

```
a[9] = a[1];
```

```
b[2] = b[2]+1;
```

```
c[26] = 't';
```

## Πίνακες

(iii)

### ◆ Διάβασμα ενός πίνακα

- γνωστό μέγεθος

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

- πρώτα διαβάζεται το μέγεθος

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

- στα παραπάνω πρέπει να προηγηθούν

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



## Πίνακες

(iv)

### ◆ Διάβασμα ενός πίνακα (συνέχεια)

- τερματισμός με την τιμή 0 (φρουρός/sentinel)  

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

```
int a[100], x;
int i;
```
- 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=0; i<10; i++) WRITELN(a[i]);
if (a[k] > a[k+1]) ...
```

### ◆ Αρχικοποίηση (με μηδενικά)

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

### ◆ Εύρεση ελάχιστου στοιχείου

```
x = a[0];
for (i=1; i<10; i++)
    if (a[i] < x) x = a[i];
```

## Γραμμική αναζήτηση

(i)

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

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

## Γραμμική αναζήτηση

(ii)

### ◆ Μια δυνατή συγκεκριμενοποίηση

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

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

## Γραμμική αναζήτηση

(iii)

### ◆ Εναλλακτική συγκεκριμενοποίηση #1

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

## Γραμμική αναζήτηση

(iv)

### ◆ Εναλλακτική συγκεκριμενοποίηση #2

```
i = 0;
do
    if (a[i] == x) found = true;
    else { found = false; i = i+1; }
while (NOT found AND i < n);
if (found)
    WRITELN("Το βρήκα στη θέση", i);
else
    WRITELN("Δεν το βρήκα");
```



## Γραμμική αναζήτηση (v)

### ◆ Εναλλακτική συγκεκριμενοποίηση #3

```
i = 0; found = false;
do
    if (a[i] == x) found = true;
    else i = i+1;
while (NOT found AND i < n);
if (found)
    WRITELN("Το βρήκα στη θέση", i);
else
    WRITELN("Δεν το βρήκα");
```

## Γραμμική αναζήτηση (vi)

### ◆ Εναλλακτική συγκεκριμενοποίηση #4

```
i = 0;
do {
    found = a[i] == x;
    i = i+1;
} while (NOT found AND i < n);
if (found)
    WRITELN("Το βρήκα στη θέση", i-1);
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

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

## Δυαδική αναζήτηση (iii)

### ◆ Αναζήτηση και εμφάνιση αποτελέσματος

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

## Πολυδιάστατοι πίνακες

### ◆ Παράδειγμα

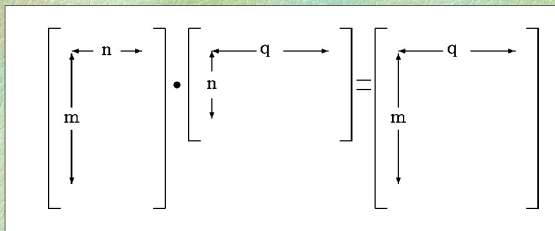
```
int a[10][16];
...
a[1][13] = 42;
...
for (i=0; i<10; i++)
    for (j=0; j<16; j++)
        a[i][j] = READ_INT();
```



## Πολλαπλασιασμός πινάκων (i)

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

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



## Πολλαπλασιασμός πινάκων (ii)

- ◆ Το πρόγραμμα

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

## 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

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

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

## Μαγικά τετράγωνα (ii)

									3											0			
																1							
																							2

## Μαγικά τετράγωνα (iii)

- ◆ Κατασκευή για περιττό  $n$

```
int a[17][17], i, j, k, h, m, n=5;
i = n/2; j = n; k = 0;
FOR (h, 1 TO n) {
  j = j-1; a[i][j] = k; k = k+1;
  FOR (m, 2 TO n) {
    j = (j+1) % n; i = (i+1) % n;
    a[i][j] = k; k = k+1;
  }
}
FOR (i, 0 TO n-1) {
  FOR (j, 0 TO n-1) WRITE(FORM(a[i][j], 4));
  WRITELN();
}
```

## 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)

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

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

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

## Αναδρομή (iii)

### ◆ Αριθμοί Fibonacci

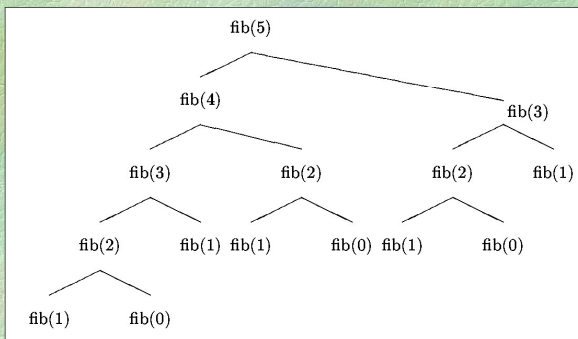
- $F_0 = 0$  ,  $F_1 = 1$
- $F_{n+2} = F_n + F_{n+1}$  ,  $\forall n \in \mathbb{N}$

### ◆ Αναδρομική συνάρτηση υπολογισμού

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

## Αναδρομή (iv)

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



## Αναδρομή (v)

### ◆ Μέγιστος κοινός διαιρέτης

- Αναδρομική υλοποίηση του αλγορίθμου του Ευκλείδη

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

## Αναδρομή (vi)

### ◆ Συνάρτηση παρόμοια με του Ackermann

$$z(i, j, 0) = j+1 \quad z(i, 0, 1) = i$$
$$z(i, 0, 2) = 0 \quad z(i, 0, n+3) = 1$$
$$z(i, j+1, n+1) = z(i, z(i, j, n+1), n) \quad , \forall i, j, n \in \mathbb{N}$$

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

## Αμοιβαία αναδρομή

```
FUNC int f2 (int n); // function prototype
```

```
FUNC int f1 (int n) {  
    if (n==0) return 5;  
    else     return f1(n-1) * f2(n-1);  
}  
  
FUNC int f2 (int n) {  
    if (n==0) return 3;  
    else     return f1(n-1) + 2*f2(n-1);  
}
```



## Αριθμητικοί υπολογισμοί (i)

### ◆ Τύπος **REAL**

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

### ◆ Παράσταση κινητής υποδιαστολής

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

## Αριθμητικοί υπολογισμοί (ii)

### ◆ Αριθμητικά σφάλματα

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

### ◆ Αναπαράσταση των αριθμών

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

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

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

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

## Εύρεση τετραγωνικής ρίζας (i)

### ◆ Χωρίς χρήση της συνάρτησης **sqrt**

### ◆ Μέθοδος Βαβυλωνίων

### ◆ Καταγράφεται για πρώτη φορά από τον Έρωνα τον Αλεξανδρέα

### ◆ Ειδική περίπτωση της γενικότερης μεθόδου του **Newton**, για την εύρεση της ρίζας οποιασδήποτε συνεχούς συναρτήσεως

$$f(y) = 0 \quad \text{εδώ: } f(y) = y^2 - x \\ \text{για κάποιο δοθέν } x$$

## Εύρεση τετραγωνικής ρίζας (ii)

### ◆ Περιγραφή της μεθόδου

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

## Εύρεση τετραγωνικής ρίζας (iii)

### ◆ Χωρίς χρήση της συνάρτησης **sqrt**

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

### ◆ Παράδειγμα: $\sqrt{37}$ (6.08276253)

$y_0 = 1$	$y_4 = 6.143246$
$y_1 = 19$	$y_5 = 6.083060$
$y_2 = 10.473684$	$y_6 = 6.082763$
$y_3 = 7.003174$	...

## Εύρεση τετραγωνικής ρίζας (iii)

```
FUNC REAL sqrt (REAL x) {
  const REAL epsilon = 0.00001; // 1E-5
  REAL older, newer = 1;

  do {
    older = newer;
    newer = (older + x/older) / 2;
  } while (NOT ( /* συνθήκη τερματισμού */ ));
  return newer;
}
```



## Εύρεση τετραγωνικής ρίζας (iv)

### ◆ Εναλλακτικές συνθήκες τερματισμού

- Σταθερός αριθμός επαναλήψεων  
`n == 20`
- Επιτυχής εύρεση ρίζας λάθος!  
`newer * newer == x`
- Απόλυτη σύγκλιση  
`abs(newer * newer - x) < epsilon`
- Σχετική σύγκλιση  
`abs(newer * newer - x) / newer < epsilon`

## Εύρεση τετραγωνικής ρίζας (v)

### ◆ Εναλλακτικές συνθήκες τερματισμού

- Απόλυτη σύγκλιση κατά Cauchy  
`abs(newer - older) < epsilon`
- Σχετική σύγκλιση  
`abs(newer - older) / newer < epsilon`

## Τριγωνομετρικές συναρτήσεις (i)

### ◆ Συνημίτονο με ανάπτυγμα Taylor

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

### ◆ για τον όρο με δείκτη $i+1$ έχουμε:

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

### ◆ οπότε αν $n = 2i+1$ έχουμε:

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

## Τριγωνομετρικές συναρτήσεις (ii)

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

## Από το PZhelp στη C++ (i)

### ◆ Τύποι δεδομένων

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

## Από το PZhelp στη C++ (ii)

<code>char , signed char , unsigned char</code>
<code>signed short int , unsigned short int</code>
<code>signed int , unsigned int</code>
<code>signed long int , unsigned long int</code>
<code>float</code>
<code>double (REAL)</code>
<code>long double</code>

Με κόκκινο χρώμα όσα μπορούν να παραλειφθούν.



## Από το PZhelp στη C++ (iii)

### ◆ Πρόγραμμα και υποπρογράμματα

<pre>int main () {     ...     // προαιρετικό     return 0; }  void p (... ) { ... }  int f (... ) { ... }</pre>	<pre>PROGRAM {     ... }  PROC p (... ) { ... }  FUNC int f (... ) { ... }</pre>
--	--

## Από το PZhelp στη C++ (iv)

### ◆ Ανάθεση

<pre>x += 42; i %= n+1;  x = y = z = 0;  y = (x = 17) + 25;  i++; /* ή */ ++i; i--; /* ή */ --i;  i = 3; x = i++; i = 3; x = ++i;  i = i++; // λάθος!</pre>	<pre>x = x + 42; i = i % (n+1);  x = 0; y = 0; z = 0; x = 17; y = x + 25;  i = i+1; i = i-1;  i = 3; x = i; i = i+1; i = 3; i = i+1; x = i;</pre>
---	---

## Από το PZhelp στη C++ (v)

### ◆ Βρόχος for

<pre>for (i=1; i&lt;=10; i++) ... for (i=8; i&gt;=1; i--) ... for (i=1; i&lt;=10; i+=2) ... // Στη C++ κανείς δε μας επιτρέπει να γράψουμε αυτό: // (πολλές κακές ιδέες μαζεμένες...) n=100; s=1; for (i=1; i&lt;=n; i+=s) {     n-=i; s++; if (i+s&gt;n) i=1; }</pre>	<pre>FOR (i, 1 TO 10) ... FOR (i, 8 DOWNTO 1) ... FOR (i, 1 TO 10 STEP 2) ... </pre>
--	--

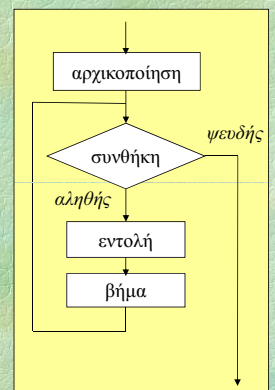
## Από το PZhelp στη C++ (vi)

### ◆ Βρόχος for

for (αρχικοποίηση; συνθήκη; βήμα)  
εντολή

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

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



## Από το PZhelp στη C++ (vii)

### ◆ Έξοδος στην οθόνη στη C++

<pre>#include &lt;iostream&gt; using namespace std; ...  cout &lt;&lt; "Hello\n"; /* ή */ cout &lt;&lt; "Hello" &lt;&lt; endl;  cout &lt;&lt; i+1; cout &lt;&lt; i &lt;&lt; " " &lt;&lt; r; cout &lt;&lt; c;</pre>	<pre>WRITELN("Hello");  WRITE(i+1); WRITE(i, r); WRITE(c);</pre>
--	--

## Από το PZhelp στη C++ (viii)

### ◆ Είσοδος από το πληκτρολόγιο στη C++

<pre>#include &lt;iostream&gt; using namespace std; ...  cin &gt;&gt; i; cin &gt;&gt; r;  cin &gt;&gt; c; /* ή */ c = cin.get();  cin.ignore(     numeric_limits&lt;streamsize&gt;::max(),     '\n'); // τρομακτικό λίγο, έτσι δεν είναι;</pre>	<pre>i = READ_INT(); r = READ_REAL();  c = getchar();  SKIP_LINE();</pre>
---	---



## Από το PZhelp στη C++ (ix)

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

```
#include <cstdio>
...

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

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

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

WRITELN("Hello");  
WRITE(i+1);  
WRITE(i, r);  
WRITE(c);  
  
WRITE(FORM(i,5));  
WRITE(FORM(r,5,3));  
  
a 97 a 97

## Από το PZhelp στη C++ (x)

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

```
#include <cstdio>
...

scanf("%d", &i);
scanf("%lf", &r);
c = getchar();
/* ή */
scanf("%c", &c);

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

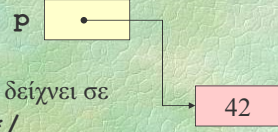
i = READ\_INT();  
r = READ\_REAL();  
c = getchar();  
  
SKIP\_LINE();

## Δείκτες (i)

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

- ◆ Παράδειγμα

```
int *p;
...
/* ο δείκτης p τοποθετείται να δείχνει σε
   κάποια ακέραια μεταβλητή */
...
*p = 42;
WRITELN(*p + 1);
```

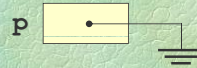


## Δείκτες (ii)

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

- ◆ Παράδειγμα

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



- ◆ Απαγορεύεται η προσπέλαση της μνήμης μέσω ενός κενού δείκτη

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

## Δείκτες (iii)

### Δεικτοδότηση: &

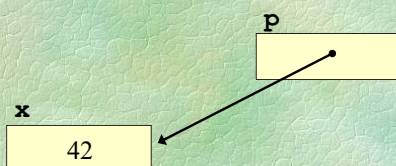
η διεύθυνση μιας μεταβλητής

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

### Αποδεικτοδότηση: \*

το περιεχόμενο μιας διεύθυνσης

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

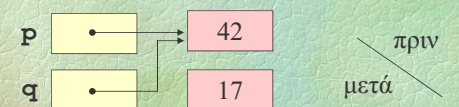


## Δείκτες και ανάθεση

- ◆ Ανάθεση δεικτών

```
q = p;
```

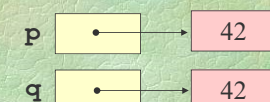
πριν



μετά

- ◆ Ανάθεση περιεχομένων

```
*q = *p;
```

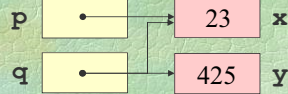


μετά



## Παράδειγμα με δείκτες

```
PROGRAM { // pointers
  int x = 42, y = 17;
  int *p, *q;
  p = &x; q = &y;
  *p = *p - *q;
  *q = *p *y;
  q = p;
  (*q)++; *p -= 3;
  WRITELN(x, y);
}
```



23 425

## Πέρασμα παραμέτρων με αναφορά

```
FUNC int gcd (int a, int b);
PROC normalize (int &p, int &q) {
  int g = gcd(p, q);
  p /= g; q /= g;
}
PROGRAM { // call_by_ref
  int x = READ_INT();
  int y = READ_INT();
  normalize(x, y);
  WRITELN(x, y);
}
```

## Δείκτες αντί περάσματος με αναφορά

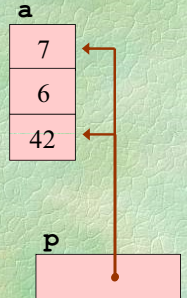
```
int gcd (int a, int b);
void normalize (int *p, int *q) {
  int g = gcd(*p, *q);
  *p /= g; *q /= g;
}
int main () {
  int x, y;
  scanf("%d %d", &x, &y);
  normalize(&x, &y);
  printf("%d %d\n", x, y);
  return 0;
}
```

Η C δεν υποστηρίζει  
πέραςμα με αναφορά!

## Πίνακες και δείκτες (i)

### Αριθμητική δεικτών

```
int a[3] = {7, 6, 42};
int *p;
p = &(a[0]);
p = &a;
p = a;
WRITELN(*p);
WRITELN(*(p+1));
p = p+2;
WRITELN(*p);
```



7  
6  
42

## Πίνακες και δείκτες (ii)

### Ισοδυναμία πινάκων και δεικτών

Ένας πίνακας είναι ένας δείκτης στο πρώτο στοιχείο.

$a[i]$  ισοδύναμο με  $*(a+i)$

Οι πίνακες όμως είναι σταθεροί δείκτες, δηλαδή δεν μπορούν να αλλάξουν τιμή

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

## Πίνακες και δείκτες (iii)

### Συμβολοσειρές

```
char a[15] = "Hello world!", b[15];
// a[12] == '\0'
void strcpy (char *t, char *s) {
  while ((*t++ = *s++) != '\0');
}
int main () {
  strcpy(b, a);
  printf("%s\n", b);
}
```



## Πίνακες και δείκτες (iv)

### Εκτύπωση συμβολοσειράς

```
void putchar (char c);

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

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

## Επεξεργασία κειμένου (i)

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

```
int n = 0;

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

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

## Επεξεργασία κειμένου (ii)

- ◆ Π.χ. αντιγραφή της εισόδου στην έξοδο

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

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

```
int c;

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

## Επεξεργασία κειμένου (iii)

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

```
int i, sum = 0;

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

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

```
int i, sum = 0;

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

## Επεξεργασία κειμένου (iv)

- ◆ Παράδειγμα 1: πρόγραμμα που

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

- ◆ Μετράμε τα **'\n'** και τους υπόλοιπους χαρακτήρες

- ◆ Ελέγχουμε για τέλος εισόδου (**EOF**)

- ◆ Για το μέσο όρο, κάνουμε διαίρεση!

## Επεξεργασία κειμένου (v)

- ◆ Παράδειγμα 1

```
int lines = 0, chars = 0;

while (true) {
    int c = getchar();
    if (c == EOF) break;
    if (c == '\n') lines++; else chars++;
}

printf("%d lines were read\n", lines);

if (lines > 0)
    printf("%0.3lf characters per line\n",
           1.0 * chars / lines);
```

- ◆ Καλύτερα: **(double) chars** μετατροπή τύπου (type cast)



## Επεξεργασία κειμένου (vi)

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

```
FUNC bool isletter (char c) {  
    return c >= 'a' AND c <= 'z'  
        OR c >= 'A' AND c <= 'Z';  
}
```

## Επεξεργασία κειμένου (vii)

- ◆ **Παράδειγμα 2**

```
int c, lines = 0, chars = 0, words = 0;  
c = getchar();  
while (c != EOF)  
    if (isletter(c)) { words++;  
        do { chars++; c = getchar(); }  
        while (isletter(c));  
    }  
    else { chars++;  
        if (c == '\n') lines++;  
        c = getchar();  
    }
```
- ◆ Έχουμε διαβάσει ένα χαρακτήρα «μπροστά»!

## Επεξεργασία κειμένου (viii)

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

## Επεξεργασία κειμένου (ix)

- ◆ **Παράδειγμα 3**

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

## Επεξεργασία κειμένου (x)

- ◆ **Παράδειγμα 3 (συνέχεια)**

```
FOR (i, 1 TO 20)  
    printf("%4d words of length %2d\n",  
        freq[i], i);
```
- ◆ Μετατροπή κεφαλαίων γραμμάτων σε πεζά

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

## Επεξεργασία κειμένου (xi)

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



## Επεξεργασία κειμένου

(xii)

### ◆ Παράδειγμα 4

```
const int MAX = 80;
int i, c, line[MAX];
while ((c = getchar()) != EOF) {
    int n = 0;
    while (c != '\n') {
        line[n++] = c; c = getchar();
    }
    FOR (i, n-1 DOWNTO 0) putchar(line[i]);
    putchar('\n');
}
```

## Επεξεργασία κειμένου

(xiii)

### ◆ Εύρεση εμφάνιση λέξης-κλειδιού

```
...
// η λέξη-κλειδί έχει 3 χαρακτήρες
FOR (j, 0 TO 2) key[j] = getchar();
...
// έστω i το μήκος της γραμμής
FOR (k, 0 TO i-3)
    if (line[k] == key[0]
        AND line[k+1] == key[1]
        AND line[k+2] == key[2])
        WRITELN("keyword found!");
```

## Συμβολοσειρές

(i)

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

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

## Συμβολοσειρές

(ii)

- ◆ Χρήσιμες συναρτήσεις βιβλιοθήκης

```
#include <cstring>
```

- ◆ Μέτρηση μήκους: `strlen`  
`printf("Your name has %d letters.\n",  
strlen(name));`
- ◆ Λεξικογραφική σύγκριση: `strcmp`  
`if (strcmp(name, "John") == 0)  
printf("I knew you were John!\n");`
- ◆ Quiz: `strcmp("ding", "dong") == ?`

## Συμβολοσειρές

(iii)

- ◆ Αντιγραφή: `strcpy`  
`char a[10];  
strcpy(a, "ding");  
a[1] = 'o';  
printf("%s\n", a); // dong`
- ◆ Συνένωση: `strcat`  
`char a[10] = "abc";  
strcat(a, "def");  
printf("%s\n", a); // abcdef`

## 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

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

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

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



## Συγκριτικές Μέθοδοι Ταξινόμησης

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

5 7 2 1 8 3 6 4

## Ταξινόμηση

(ii)

- ◆ Ταξινόμηση με επιλογή (selection sort)

```
FOR (i, 0 TO n-2) {
    int minpos = i;
    FOR (j, i+1 TO n-1)
        if (a[j] < a[minpos]) minpos = j;
    swap(a[i], a[minpos]);
}
```

- ◆ Πλήθος συγκρίσεων;

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

5 7 2 1 8 3 6 4

## Ταξινόμηση

(iii)

- ◆ Ταξινόμηση με εισαγωγή (insertion sort)

```
FOR (i, 1 TO n-1) {
    int x = a[i], j = i;
    while (j > 0 AND a[j-1] > x) {
        a[j] = a[j-1]; j = j-1;
    }
    a[j] = x;
}
```

5	7	2	1	8	3	6	4	$i=1$	1	2	5	7	8	3	6	4	$i=5$
5	7	2	1	8	3	6	4	$i=2$	1	2	3	5	7	8	6	4	$i=6$
2	5	7	1	8	3	6	4	$i=3$	1	2	3	5	6	7	8	4	$i=7$
1	2	5	7	8	3	6	4	$i=4$	1	2	3	4	5	6	7	8	

## Ταξινόμηση

(iv)

- ◆ Ταξινόμηση με εισαγωγή (insertion sort)

```
FOR (i, 1 TO n-1) {
    int x = a[i], j = i;
    while (j > 0 AND a[j-1] > x) {
        a[j] = a[j-1]; j = j-1;
    }
    a[j] = x;
}
```

- ◆ Πλήθος συγκρίσεων;

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

## Ταξινόμηση

(v)

- ◆ Μέθοδος της φυσαλίδας (bubble sort)

```
FOR (i, 0 TO n-2)
    FOR (j, n-2 DOWNT0 i)
        if (a[j] > a[j+1])
            swap(a[j], a[j+1]);
```

- ◆ Πλήθος συγκρίσεων

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

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

## Ταξινόμηση

(vi)

- ◆ Παράδειγμα εκτέλεσης ( $n = 7$ )

input: 12 4 9 8 6 7 5

12 4 9 8 6 5 7  
 12 4 9 8 5 6 7  
 12 4 9 5 8 6 7  
 12 4 5 9 8 6 7  
 12 4 5 9 8 6 7  
 $i = 0$  4 12 5 9 8 6 7

4 12 5 9 8 6 7  
 4 12 5 9 6 8 7  
 4 12 5 6 9 8 7  
 4 12 5 6 9 8 7  
 $i = 1$  4 5 12 6 9 8 7

4 5 12 6 9 7 8  
 4 5 12 6 7 9 8  
 $i = 2$  4 5 6 12 7 9 8

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

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

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



## Ταξινόμηση (vii)

### ◆ Βελτίωση με έλεγχο εναλλαγών

```
FOR (i, 0 TO n-2) {  
  bool swaps = false;  
  FOR (j, n-2 DOWNTO i)  
    if (a[j] > a[j+1]) {  
      swaps = true;  
      swap(a[j], a[j+1]);  
    }  
  if (NOT swaps) break;  
}
```

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

## Ταξινόμηση (viii)

### ◆ Ταξινόμηση με συγχώνευση (merge sort)

- Διαιρώ την ακολουθία των αριθμών σε δύο μέρη
- Με αναδρομικές κλήσεις, ταξινομώ τα δύο μέρη ανεξάρτητα
- Συγχωνεύω τα δύο ταξινομημένα μέρη

### ◆ Στη χειρότερη περίπτωση απαιτούνται $O(n \log n)$ συγκρίσεις

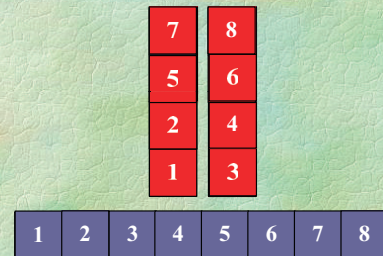
## Ταξινόμηση (ix)

### ◆ Ταξινόμηση με συγχώνευση

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

## Ταξινόμηση (x)

### ◆ Συγχώνευση



## Ταξινόμηση (xi)

### ◆ Συγχώνευση

```
PROC merge (int a[], int first,  
           int mid, int last) {  
  int b[last-first+1];  
  int i = first, j = mid+1, k = 0;  
  while (i <= mid AND j <= last)  
    if (a[i] < a[j]) b[k++] = a[i++];  
    else b[k++] = a[j++];  
  while (i <= mid) b[k++] = a[i++];  
  while (j <= last) b[k++] = a[j++];  
  FOR (i, 0 TO k-1) a[first+i] = b[i];  
}
```

## Ταξινόμηση (xii)

### ◆ Διαδοχικές ταξινομήσεις και συγχωνεύσεις





## Ταξινόμηση

(xiii)

### ◆ Ταξινόμηση με συγχώνευση (διαφορετικά)

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

## Ταξινόμηση

(xiv)

### ◆ Συγχώνευση (διαφορετικά)

```
PROC merge (int *first, int *mid,
            int *last) {
    int b[last-first];
    int *i = first, *j = mid, *k = b;
    while (i < mid AND j < last)
        if (*i < *j) *k++ = *i++;
        else *k++ = *j++;
    while (i < mid) *k++ = *i++;
    while (j < last) *k++ = *j++;
    i = first; j = b;
    while (j < k) *i++ = *j++;
}
```

## Ταξινόμηση

(xv)

### ◆ Ταξινόμηση με διαμέριση (quick sort)

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

## Ταξινόμηση

(xvi)

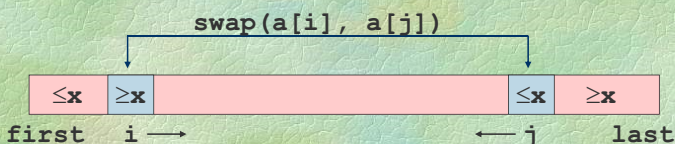
### ◆ Διαμέριση (partition)

```
FUNC int partition (int a[], int first,
                   int last) {
    // επιλογή ενός στοιχείου
    int x = a[(first + last)/2];
    int i = first, j = last;
    while (true) {
        while (a[i] < x) i++;
        while (x < a[j]) j--;
        if (i >= j) break;
        swap(a[i], a[j]); i++; j--;
    }
    return j;
}
```

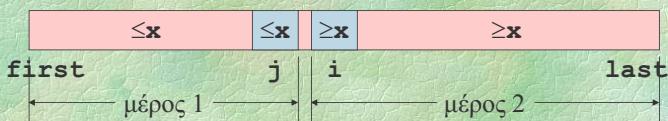
## Ταξινόμηση

(xvii)

### ◆ Σε κάθε βήμα της διαμέρισης



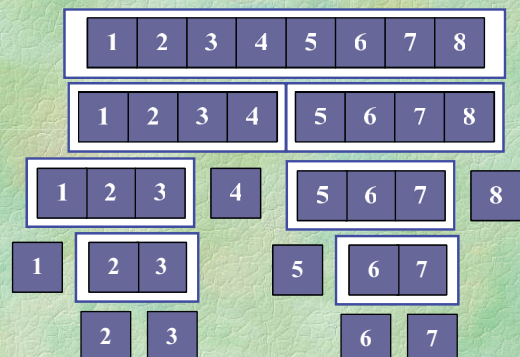
### ◆ Μετά τη διαμέριση



## Ταξινόμηση

(xviii)

### ◆ Διαδοχικές διαμερίσεις και ταξινομήσεις

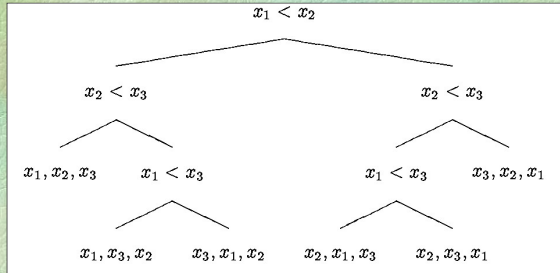




## Ταξινόμηση

(xix)

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



## Ορθότητα

(i)

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

## Ορθότητα

(ii)

- ◆ Παράδειγμα: εύρεση γινομένου

```
FUNC int mult (int x, int y) {  
    int i, z = 0;  
    FOR (i, 1 TO x) z = z+y;  
    return z;  
}
```

- ◆ Ισχυρισμός:

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

## Ορθότητα

(iii)

- ◆ Εντοπισμός σημείων όπου θα γραφούν βεβαιώσεις

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



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



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

## Ορθότητα

(v)

- ◆ Παράδειγμα: υπολογισμός δύναμης με επαναλαμβανόμενο τετραγωνισμό (Gauss)

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



## Ορθότητα

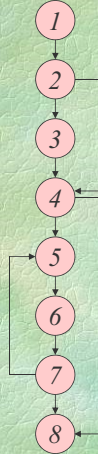
(vi)

### ◆ Ροή ελέγχου

### ◆ Βεβαιώσεις

```

/*1 - Βεβαίωση εισόδου: REAL y, int j */
/*2 : x = y, i = j */
/*3 : i < 0 */
/*4 : i ≥ 0, yi = xi */
/*5 - Αναλλοίωτη βρόχου: i ≥ 0, yi = z * xi */
/*6 : i ≥ 0, yi = z * xi αν i άρτιος,
      yi = z * xi-1 αν i περιττός */
/*7 : yi = z * xi */
/*8 - Βεβαίωση εξόδου: yi = z * xi */
    
```



## 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

## Τεχνολογία λογισμικού

### ◆ Software engineering

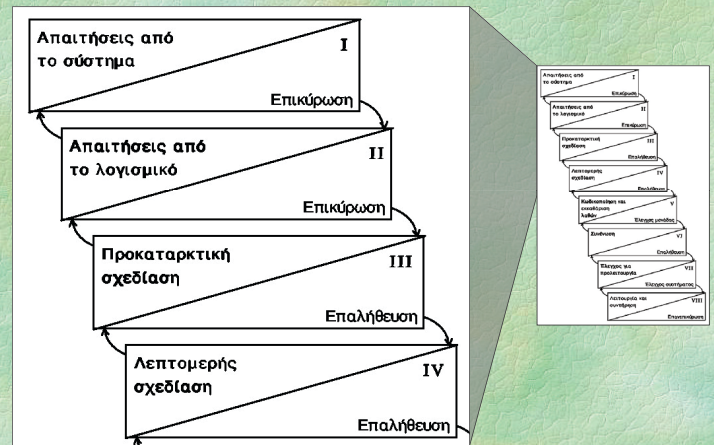
### ◆ Ανάπτυξη λογισμικού που να εξασφαλίζει:

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

### ◆ Μοντέλα κύκλου ζωής λογισμικού

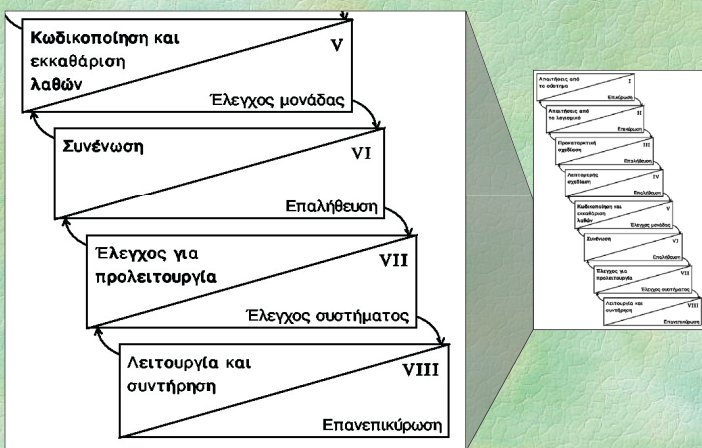
## Μοντέλο του καταρράκτη

(i)



## Μοντέλο του καταρράκτη

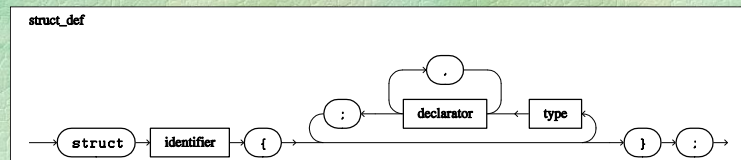
(ii)



## Δομές

(i)

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





## Δομές

(ii)

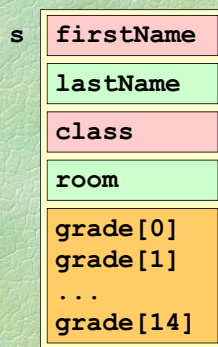
### ◆ Παράδειγμα

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

student s;

...

s.class = 3;
WRITELN(s.firstName, s.lastName);
```



## Δομές

(iii)

### ◆ Παράδειγμα: μέσος όρος βαθμολογίας

```
FUNC REAL average (student s) {
    REAL sum = 0.0;
    int i;

    FOR (i, 0 TO 14) sum += s.grade[i];
    return sum / 15;
}
```

## Δομές μέσα σε δομές

(iv)

```
struct date {
    int day, month, year;
};

struct student {
    ...
    date birthDate;
    ...
};

...

WRITELN(s.birthDate.day, "/",
        s.birthDate.month, "/",
        s.birthDate.year);
```

## Τι είναι οι μιγαδικοί αριθμοί;

say  
what!

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

$$i^2 = -1$$

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

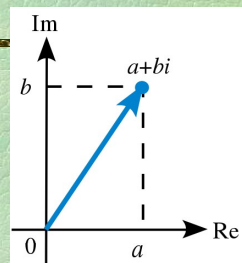
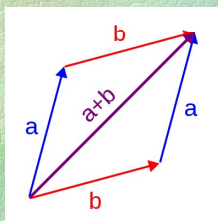
## Μιγαδικοί αριθμοί

$a$ : πραγματικό μέρος

$b$ : φανταστικό μέρος

Αριθμητικές πράξεις, π.χ.

πρόσθεση



Για περισσότερα, ρωτήστε το μαθηματικό σας!

## Μιγαδικοί αριθμοί

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

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

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

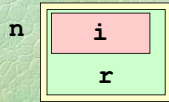


## Ενώσεις

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

◆ **Παράδειγμα**

```
union number { int i; REAL r; };
number n;
n.r = 1.2;
printf("%lf\n", n.r);
n.i = 42;
printf("%d\n", n.i);
printf("%lf\n", n.r); /* λάθος! */
```



## Αρχεία

(i)

◆ **Αρχείο (file):** αποτελείται από

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

◆ **Παράδειγμα**

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

## Αρχεία

(ii)

◆ **Άνοιγμα και κλείσιμο αρχείου**

fopen fclose

◆ **Διάβαση και γράψιμο**

fputc	fgetc	χαρακτήρες
fputs	fgets	συμβολοσειρές
fprintf	fscanf	οτιδήποτε
fwrite	fread	ακολουθίες byte

◆ **Έλεγχος τέλους αρχείου**

feof

## Αρχεία κειμένου στη C++

(i)

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

```
#include <cstdio>
int main (int argc, char *argv[]) {
    int i;
    for (i=1; i<argc; i++)
        // επεξεργασία του αρχείου argv[i]
}
```

## Αρχεία κειμένου στη C++

(ii)

◆ **Παράδειγμα (συνέχεια)**

```
// επεξεργασία του αρχείου argv[i]
FILE *f;
int chars = 0, lines = 0, c;
if ((f = fopen(argv[i], "rt")) == nullptr)
    return 1;
while ((c = fgetc(f)) != EOF) {
    chars++;
    if (c == '\n') lines++;
}
fclose(f);
printf("%d chars, %d lines, %s\n",
    chars, lines, argv[i]);
```

## Δυαδικά αρχεία στη C++

(i)

◆ **Παράδειγμα: αντιγραφή δυαδικών αρχείων**

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



## Διαδικά αρχεία στη C++ (ii)

### ◆ Παράδειγμα (συνέχεια)

```
while (!feof(fin)) {
    unsigned char buffer[1000];
    unsigned int count;

    count = fread(buffer, 1, 1000, fin);
    fwrite(buffer, 1, count, fout);
}

fclose(fin);
fclose(fout);
}
```

## 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: $O$

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

### ◆ Κάτω φράγμα: $\Omega$

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

### ◆ Τάξη μεγέθους: $\Theta$

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

- Γράφουμε  $g = O(f)$  αντί  $g \in O(f)$
- π.χ.  $5n^2 + 4n - 2n \log n + 7 = \Theta(n^2)$

## Πολυπλοκότητα (iii)

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

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

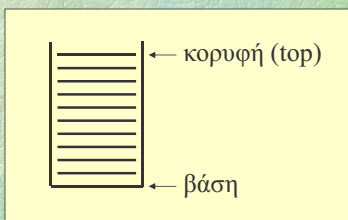
$2^{2^n}$  η υπερεκθετική συνάρτηση:  $2^{2^{\dots^2}}$  ( $n$  φορές)  
και  $\log^* n$  η αντίστροφή της

$A(n)$  η συνάρτηση Ackermann και  $\alpha(n)$  η αντίστροφή της

## Στοιβες (i)

### ◆ Last In First Out (LIFO)

ό,τι μπαίνει τελευταίο, βγαίνει πρώτο



## Στοιβες (ii)

### ◆ Αφηρημένος τύπος δεδομένων

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

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



## Στοιβες

(iii)

- ◆ Αντικείμενα (objects) στη C++  
Υλοποίηση στοιβάς με πίνακα

```
class stack {  
public:  
    stack ();  
    bool empty ();  
    void push (int x);  
    int pop ();  
private:  
    const int size = 100;  
    int data[size];  
    int top;  
};
```

κατασκευαστής (constructor)

μέθοδοι (methods)

πεδία (fields)  
όπως στο struct

## Στοιβες

(iv)

- ◆ Παράδειγμα χρήσης

```
PROGRAM { // stack_demo  
  
    stack s; // καλείται ο κατασκευαστής  
    int i;  
  
    FOR (i, 1 TO 10) s.push(i);  
  
    while (!s.empty())  
        WRITELN(s.pop());  
}
```

## Στοιβες

(v)

- ◆ Κατασκευή άδειας στοιβάς

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

- ◆ Έλεγχος αν μια στοιβα είναι άδεια

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

## Στοιβες

(vi)

- ◆ Προσθήκη στοιχείου

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

- ◆ Αφαίρεση στοιχείου

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

## Διαχείριση της μνήμης

(i)

- ◆ Στατικές μεταβλητές: γενικές ή τοπικές

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

- ◆ Δυναμικές μεταβλητές

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

## 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)

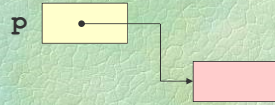


## Δυναμική παραχώρηση μνήμης (i)

### ◆ Δέσμευση (C++)

- δημιουργία μιας νέας δυναμικής μεταβλητής

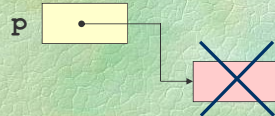
```
int *p;  
...  
p = new int;
```



### ◆ Αποδέσμευση

- καταστροφή μιας δυναμικής μεταβλητής

```
delete p;
```

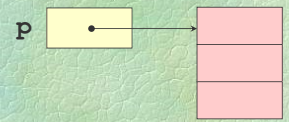


## Δυναμική παραχώρηση μνήμης (ii)

### ◆ Δέσμευση

- δημιουργία πίνακα μεταβλητού μεγέθους

```
int *p, n;  
...  
n = 3;  
p = new int[n];
```



// αν δεν υπάρχει αρκετή μνήμη, προκύπτει **εξαιρέση**  
// σε αυτό το μάθημα δε θα μιλήσουμε για εξαιρέσεις στη C++

### ◆ Αποδέσμευση

```
delete [] p;
```

## Δυναμική παραχώρηση μνήμης (iii)

### ◆ Δέσμευση και αποδέσμευση σε C

```
#include <stdlib.h>    p = new int[42];  
...                   ...  
int *p;                delete [] p;  
...                   ...
```

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

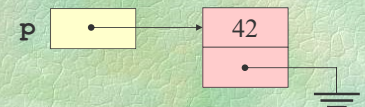
## Σύνθετες δυναμικές μεταβλητές (i)

### ◆ Παράδειγμα

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

```
thing *p;
```

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

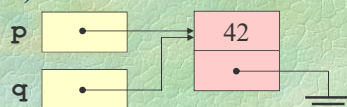


**p->something** ισοδύναμο με **(\*p).something**

## Σύνθετες δυναμικές μεταβλητές (ii)

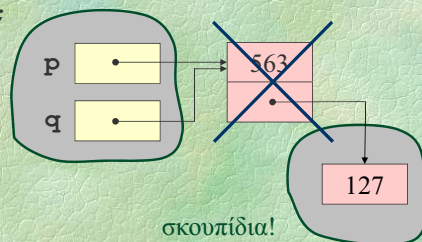
### ◆ Παράδειγμα (συνέχεια)

```
q = p;  
q->num = 563;  
q->ptr = new int;  
*(q->ptr) = 127;
```



```
delete p;
```

ξεκρέμαστοι δείκτες!



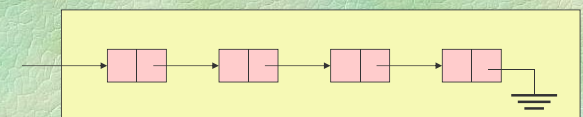
## Συνδεδεμένες λίστες (i)

### ◆ Είναι γραμμικές διατάξεις

### ◆ Κάθε κόμβος περιέχει:

- κάποια πληροφορία
- ένα σύνδεσμο στον επόμενο κόμβο

### ◆ Ο τελευταίος κόμβος έχει κενό σύνδεσμο



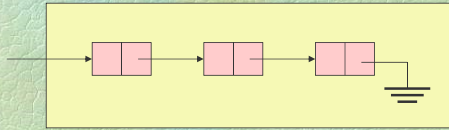


## Συνδεδεμένες λίστες

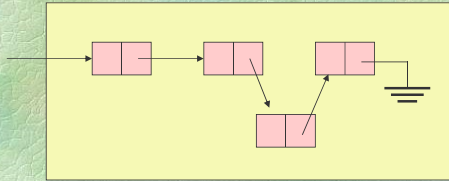
(ii)

### ◆ Ευκολότερη προσθήκη στοιχείων

- πριν



- μετά

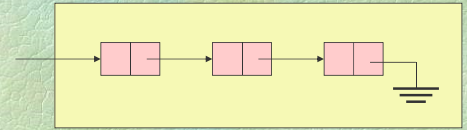


## Συνδεδεμένες λίστες

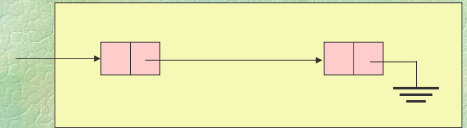
(iii)

### ◆ Ευκολότερη διαγραφή στοιχείων

- πριν



- μετά



## Συνδεδεμένες λίστες

(iv)

### ◆ Τύπος κόμβου συνδεδεμένης λίστας

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

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

```
node *head;
```

## Συνδεδεμένες λίστες

(v)

### ◆ Παράδειγμα κατασκευής λίστας

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

## Συνδεδεμένες λίστες

(vi)

### ◆ Εκτύπωση λίστας

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

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

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

## Συνδεδεμένες λίστες

(vii)

### ◆ Εκτύπωση λίστας με αναδρομή

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

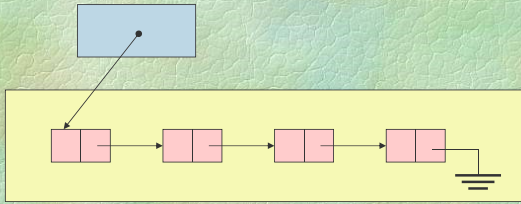
### ◆ Εκτύπωση λίστας αντίστροφα με αναδρομή

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



## Στοιβες (ξανά) (i)

- ◆ Υλοποίηση με απλά συνδεδεμένη λίστα



## Στοιβες (ξανά) (ii)

- ◆ Υλοποίηση με απλά συνδεδεμένη λίστα

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

ίδια όπως πριν!

## Στοιβες (ξανά) (iii)

- ◆ Κατασκευή άδειας στοιβάς

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

- ◆ Έλεγχος αν μια στοιβά είναι άδεια

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

## Στοιβες (ξανά) (iv)

- ◆ Προσθήκη στοιχείου

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

## Στοιβες (ξανά) (v)

- ◆ Αφαίρεση στοιχείου

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

## Στοιβες (ξανά) (vi)

- ◆ Παράδειγμα χρήσης

```
PROGRAM { // stack_demo  
  
    stack s; // καλείται ο κατασκευαστής  
    int i;  
  
    FOR (i, 1 TO 10) s.push(i);  
  
    while (!s.empty())  
        WRITELN(s.pop());  
}
```

Δεν άλλαξε τίποτα!

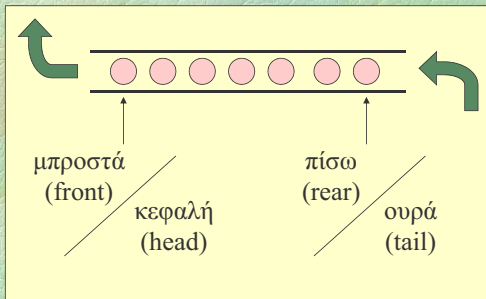


## Ουρές

(i)

### ◆ First In First Out (FIFO)

ό,τι μπαίνει πρώτο, βγαίνει πρώτο



## Ουρές

(ii)

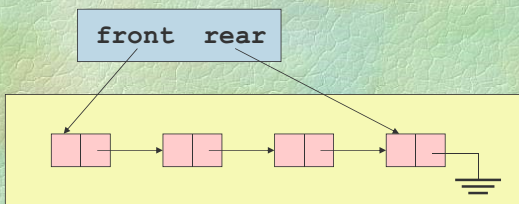
### ◆ Αφηρημένος τύπος δεδομένων

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

## Ουρές

(iii)

### ◆ Υλοποίηση με απλά συνδεδεμένη λίστα



## Ουρές

(iv)

### ◆ Υλοποίηση με απλά συνδεδεμένη λίστα

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

## Ουρές

(v)

### ◆ Κατασκευή άδειας ουράς

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

### ◆ Έλεγχος αν μια ουρά είναι άδεια

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

## Ουρές

(vi)

### ◆ Προσθήκη στοιχείου

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



## Ουρές

(vii)

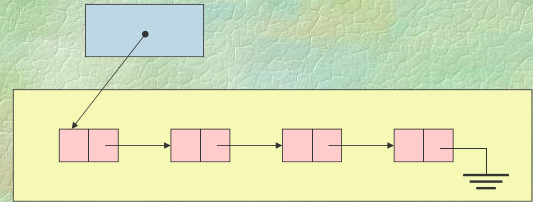
### ◆ Αφαίρεση στοιχείου

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

## Γραμμικές λίστες

(i)

### ◆ Γενική μορφή απλά συνδεδεμένης λίστας



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

## Γραμμικές λίστες

(ii)

### ◆ Εισαγωγή στο τέλος

$O(n)$

```
PROC insertAtRear (list &l, int data) {
    node *p, *q;

    p = new node;
    p->info = data; p->next = nullptr;
    if (l == nullptr) l = p;
    else {
        q = l;
        while (q->next != nullptr)
            q = q->next;
        q->next = p;
    }
}
```

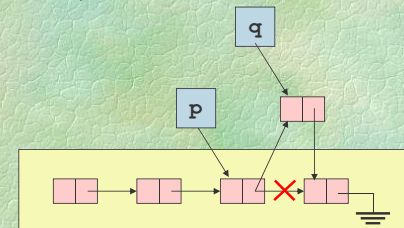
## Γραμμικές λίστες

(iii)

### ◆ Εισαγωγή μετά τον κόμβο p

$O(1)$

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



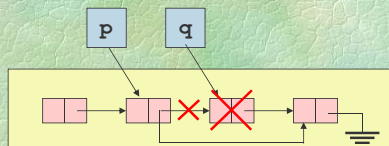
## Γραμμικές λίστες

(iv)

### ◆ Διαγραφή του κόμβου μετά τον p

$O(1)$

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



## Γραμμικές λίστες

(v)

### ◆ Εύρεση στοιχείου

$O(n)$

```
FUNC node *search (list l, int data) {
    node *p;

    for (p = l; p != nullptr; p = p->next)
        if (p->info == data) return p;
    return nullptr;
}
```



## Γραμμικές λίστες

(vi)

### ◆ Αντιστροφή λίστας

$O(n)$

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

## Γραμμικές λίστες

(vii)

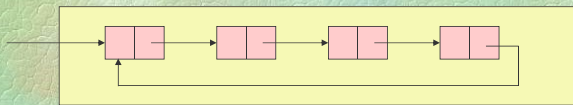
### ◆ Συνένωση δύο λιστών

$O(n)$

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

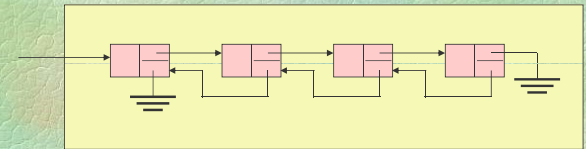
## Κυκλικές λίστες

- ◆ Ο επόμενος του τελευταίου κόμβου είναι πάλι ο πρώτος



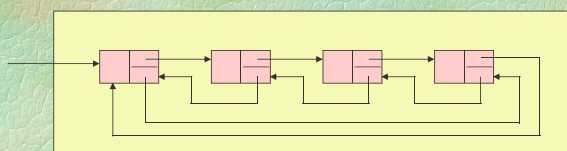
## Διπλά συνδεδεμένες λίστες

- ◆ 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)

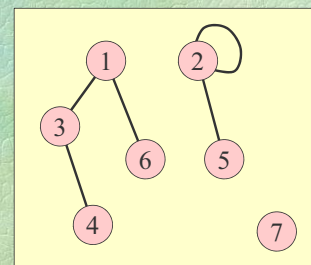
- ◆ Γράφος ή γράφημα (graph)  $G = (V, E)$

- $V$  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



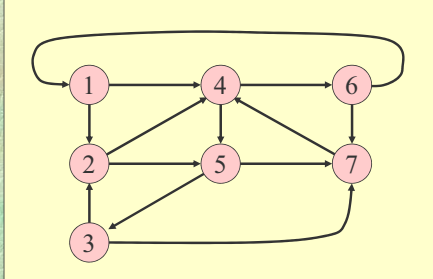


## Γράφοι

(ii)

### ◆ Κατευθυνόμενος γράφος (directed graph)

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

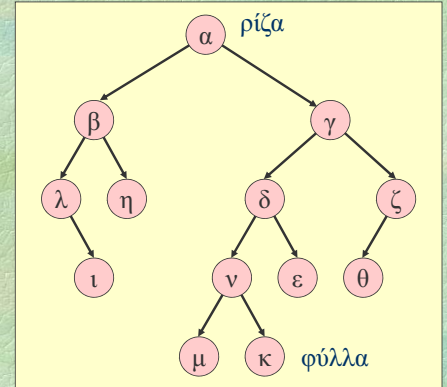


## Δυαδικά δέντρα

(i)

### ◆ Ειδικοί γράφοι της μορφής:

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

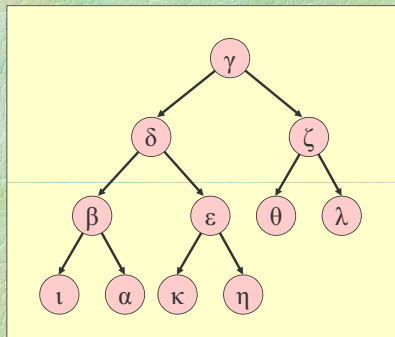


## Δυαδικά δέντρα

(ii)

### ◆ Πλήρες δυαδικό δέντρο:

- ◆ Μόνο το κατώτατο επίπεδο μπορεί να μην είναι πλήρες



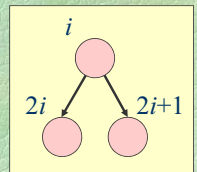
- ◆ Πλήθος κόμβων =  $n \Rightarrow$  ύψος =  $O(\log n)$

## Δυαδικά δέντρα

(iii)

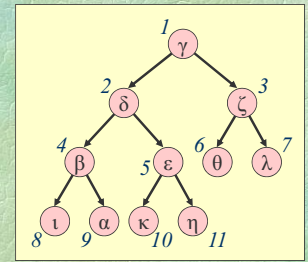
### ◆ Υλοποίηση με πίνακα

- 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'
```

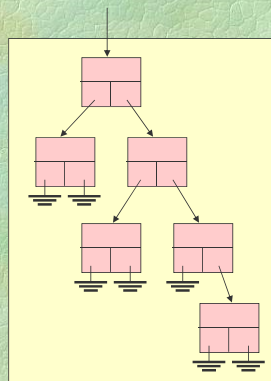
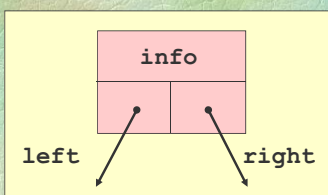


## Δυαδικά δέντρα

(iv)

### ◆ Υλοποίηση με δείκτες

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



## Δυαδικά δέντρα

(v)

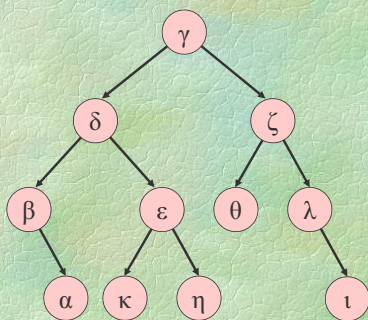
### ◆ Διάσχιση όλων των κόμβων ενός δένδρου

- 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



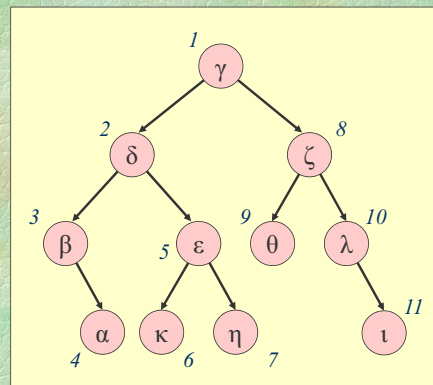
## Δυαδικά δέντρα (vi)

### ◆ Διάσχιση preorder / inorder / postorder



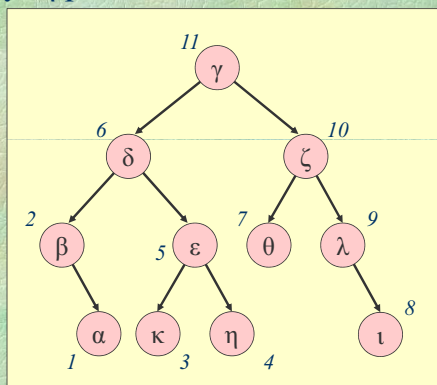
## Δυαδικά δέντρα (vi)

### ◆ Διάσχιση preorder



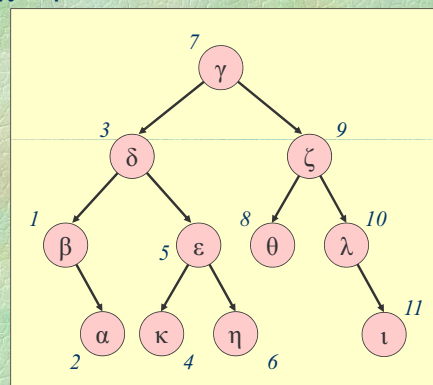
## Δυαδικά δέντρα (vii)

### ◆ Διάσχιση postorder



## Δυαδικά δέντρα (viii)

### ◆ Διάσχιση inorder



## Δυαδικά δέντρα (ix)

### ◆ Υλοποίηση της διάσχισης preorder

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

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

## Δυαδικά δέντρα (x)

### ◆ Πλήθος κόμβων και ύψος δέντρου

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

Πολυπλοκότητα:  
 $O(n)$

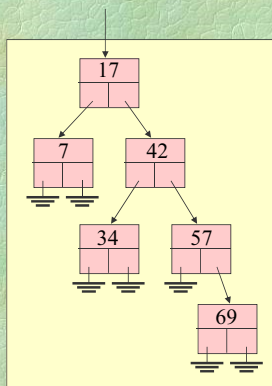


## Δυαδικά δέντρα αναζήτησης (i)

### ◆ Binary search trees

#### ◆ Για κάθε κόμβο ισχύουν οι παρακάτω ιδιότητες:

- όλοι οι κόμβοι του αριστερού υποδέντρου έχουν τιμές **μικρότερες ή ίσες** της τιμής του κόμβου
- όλοι οι κόμβοι του δεξιού υποδέντρου έχουν τιμές **μεγαλύτερες ή ίσες** της τιμής του κόμβου



## Δυαδικά δέντρα αναζήτησης (ii)

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

## Δυαδικά δέντρα αναζήτησης (iii)

### ◆ Αναζήτηση

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

## Δυαδικά δέντρα αναζήτησης (iv)

### ◆ Εισαγωγή

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

## Δυαδικά δέντρα αναζήτησης (v)

### ◆ Ισοζύγισμα

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

## Το λειτουργικό σύστημα Unix (i)

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



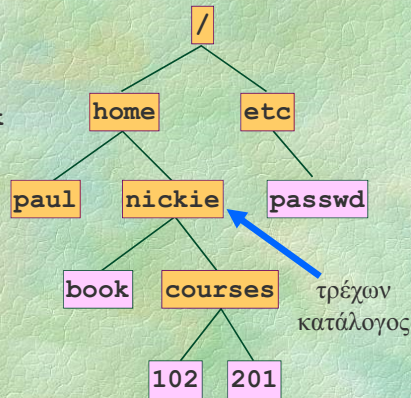
## Το λειτουργικό σύστημα Unix (ii)

### ◆ Απόλυτα ονόματα

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

### ◆ Σχετικά ονόματα

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



## Το λειτουργικό σύστημα Unix (iii)

### ◆ Θετικά στοιχεία του Unix

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

### ◆ Αρνητικά στοιχεία του Unix

- κρυπτογραφικά ονόματα εντολών
- περιορισμένη και συνθηματική βοήθεια

## Σύστημα αρχείων του Unix (i)

### ◆ Αντιγραφή αρχείων

**cp**

```
cp oldfile newfile
cp 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**
- Ονομα **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.
  - Επικράτειες (domains) στο δίκτυο της Ελλάδας

## 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$
```



## ◆ Μεταφορά αρχείων (FTP)

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

## ◆ Ηλεκτρονικά νέα (news)

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

## ◆ Κουτσομπολιό (chat ή IRC)

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

◆ Παγκόσμιος ιστός  
World-Wide Web (WWW)

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

## ◆ Διευθύνσεις στον παγκόσμιο ιστό (URL)

`http://www.corelab.ece.ntua.gr/courses/programming/`

↓  
είδος  
πληροφορίας  
και πρωτόκολλο  
επικοινωνίας
↓  
όνομα  
εξυπηρετητή
↓  
θέση στον  
εξυπηρετητή

## ◆ Παραδείγματα διευθύνσεων

`http://www.ntua.gr/`  
`ftp://ftp.ntua.gr/pub/linux/README.txt`  
`news://news.ntua.gr/comp.lang.pascal`