

Βασικές Εξειδικεύσεις σε Θεωρία και Λογισμικό



Η γλώσσα προγραμματισμού C

Ακ. Έτος 2004-2005, τμήμα Α2

Βασίλειος Βεσκούκης,
Δρ. Μηχανικός ΕΜΠ

Η γλώσσα προγραμματισμού C



Διαφάνειες διαλέξεων
Σχολής ΗΜΜΥ ΕΜΠ

Νίκος Παπασπύρου, Λέκτορας ΕΜΠ
Βασίλης Βεσκούκης, Επισκ.Επ.Καθ.ΕΜΠ

Η ιστορία της C

- 1969-1973** AT&T Bell Labs, Dennis Ritchie
- 1978** "The C Programming Language"
K&R: Kernighan & Ritchie
- 1983** Σύσταση ANSI Standardization Committee X3J11
- 1989-1990** Αποδοχή ANSI/ISO Standard
⇒ ANSI C
- 1990-1999** Αναθεώρηση του standard υπό εξέλιξη — C9X
⇒ C99

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Χαρακτηριστικά της C (i)

- ☐ Γλώσσα προστακτικού προγραμματισμού
- ☐ Γλώσσα μετρίου επιπέδου
- ☐ Οικονομία στην έκφραση (λιτή και περιεκτική)
- ☐ Σχετικά χαλαρό σύστημα τύπων
- ☐ Φιλοσοφία: ο προγραμματιστής έχει πλήρη έλεγχο και ευθύνεται για τα σφάλματά του

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Χαρακτηριστικά της C (ii)

- ❑ Ιδιαίτερα δημοφιλής στην πράξη
- ❑ Έχει χρησιμοποιηθεί για τον προγραμματισμό ευρέως φάσματος συστημάτων και εφαρμογών
- ❑ Έχει χρησιμοποιηθεί ως βάση για πληθώρα άλλων γλωσσών: **C++, Java**

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Hello world!

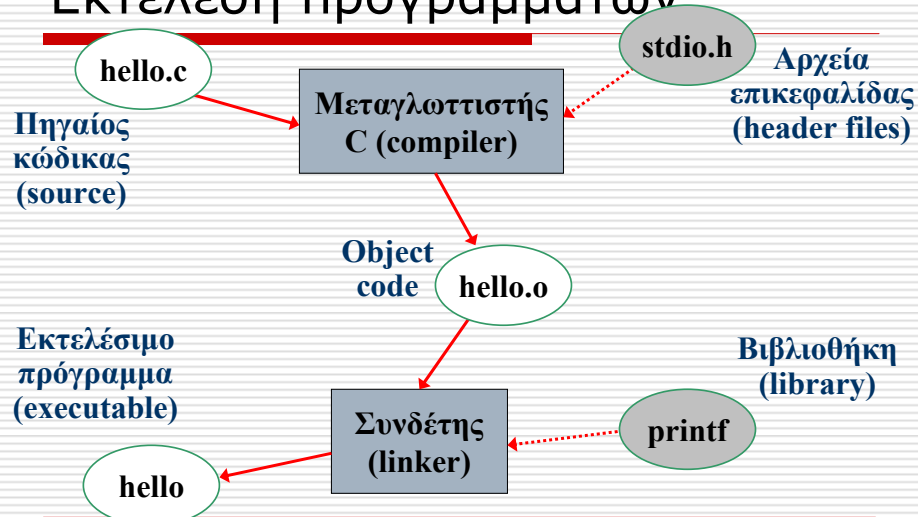
```
#include <stdio.h>

void main ()
{
    printf("Hello world!\n");
}
```

- ❑ Παρατηρήσεις
 - Η επικοινωνία με τον «έξω» κόσμο γίνεται μέσω της **βιβλιοθήκης συναρτήσεων**
 - Το σημείο έναρξης του προγράμματος είναι η «κύρια» συνάρτηση **main**.

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Εκτέλεση προγραμμάτων



Τύποι δεδομένων (απλοί)

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

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Απλοί τύποι δεδομένων

char , signed char , unsigned char
signed short **int** , unsigned short **int**
signed int , unsigned int
signed long **int** , unsigned long **int**
float
double
long double

□ Με κόκκινο χρώμα όσα μπορούν να παραλειφθούν.

N.Πασισπύρου, Β.Βεσκούκης, ΕΜΠ

Ορισμός μεταβλητών

```
int x;  
int x, y, z;  
double r;  
unsigned long abc;
```

□ Αρχικοποίηση

```
int x = 1;  
int x, y = 0, z = 2;  
double r = 1.87;  
unsigned long abc = 42000000;
```

N.Πασισπύρου, Β.Βεσκούκης, ΕΜΠ

Σταθερές (i)

□ Ακέραιες

42	0	-1	δεκαδικές
037			οκταδικές
0x1f			δεκαεξαδικές
42U	42L	42UL	unsigned & long

□ Κινητής υποδιαστολής

42.0	-1.3	δεκαδικές
2.99e8		με δύναμη του 10
42.0F	42.0L	float & long double

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Σταθερές (ii)

□ Χαρακτήρα

'a'	'0'	'\$'
-----	-----	------

□ Ειδικοί χαρακτήρες

\n	αλλαγή γραμμής
\'	απόστροφος
\\	χαρακτήρας \ (backslash)
\t	αλλαγή στήλης (tab)
\"	εισαγωγικό
\0	χαρακτήρας με ASCII = 0 (null)
\037	» με ASCII = 37 (οκταδικό)
\x1f	» με ASCII = 1f (δεκαεξαδικό)

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Σταθερές (iii)

❑ Συμβολοσειρές

```
"abc" "Hello world!\n" "a\"51\""
```

❑ Δηλώσεις σταθερών

```
const int size = 10, num = 5;  
const double pi = 3.14159;  
const char newline = '\n';
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Σχόλια

❑ Μεταξύ /* και */

```
#include <stdio.h>  
/* This simple program greets the  
world by saying "hello" */  
void main ()  
{  
    printf( /* eh? */ "Hello world!\n");  
}
```

❑ Το παρακάτω είναι λάθος

```
/* Nested /* comments */ are wrong! */
```



N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Εκτύπωση με την printf

□ Απλοί τύποι δεδομένων

■ int	%d
■ char	%c
■ double	%lf
■ string	%s

□ Παράδειγμα

```
printf("%d %lf %c %s\n",  
      42, 1.2, 'a', "hello");
```

□ Αποτέλεσμα

```
42 1.200000 a hello
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Εισαγωγή με την scanf

□ Ίδιοι κωδικοί για τους απλούς τύπους

□ Παράδειγμα

```
int n;  
double d;  
char c;  
scanf("%d", &n);  
scanf("%lf", &d);  
scanf("%c", &c);
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ένα πιο σύνθετο παράδειγμα

```
#include <stdio.h>

void main ()
{
    int celcius;
    double fahrenheit;

    printf("Give the temperature (C): ");
    scanf("%d", &celcius);
    fahrenheit = 9.0 * celcius / 5.0 + 32.0;
    printf("%d degrees Celcius "
           "is %lf degrees Fahrenheit",
           celcius, fahrenheit);
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Τελεστές και εκφράσεις (i)

□ Αριθμητικοί τελεστές

+ - * / %

□ Σχεσιακοί τελεστές

== != < > <= >=

□ Λογικοί τελεστές

&& λογική σύζευξη (και)

|| λογική διάζευξη (ή)

! λογική άρνηση (όχι)

□ Π.Χ. (x % 3 != 0) && !finished

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Τελεστές και εκφράσεις (ii)

□ Τελεστές bit προς bit (bitwise)

&	σύζευξη bit	(AND)
	διάζευξη bit	(OR)
^	αποκλειστική διάζευξη bit	(XOR)
~	άρνηση	(NOT)
<<	ολίσθηση bit αριστερά	
>>	ολίσθηση bit δεξιά	

□ Παράδειγμα

(0x0101 & 0xffff0) << 2
⇒ 0x0400

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Τελεστές και εκφράσεις (iii)

■ Τελεστής συνθήκης

(a >= b) ? a : b

■ Τελεστής παράθεσης

a-1, b+5

■ Τελεστές ανάθεσης

a = b+1

a += x ισοδύναμο με **a = a + x**

■ Τελεστές αύξησης και μείωσης

a++ **a--** τιμή πριν τη μεταβολή

++a **--a** τιμή μετά τη μεταβολή

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Εντολές και έλεγχος ροής (i)

- Κενή εντολή
;
- Εντολή έκφρασης
a = b+5;
a++;
- Εντολή if
if (a >= b)
 max = a;
else
 max = b;

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Εντολές και έλεγχος ροής (ii)

- Σύνθετη εντολή
if (a >= b) {
 min = b;
 max = a;
}
else {
 max = b;
 min = a;
}
- Ορίζει νέα
 εμβέλεια
if (x < y) {
 int temp = x;
 x = y;
 y = temp;
}

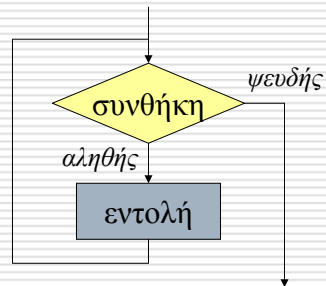
N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Εντολές και έλεγχος ροής (iii)

while (συνθήκη)
 εντολή

□ Εντολή while

```
int i = 1, s = 0;  
while (i <= 10) {  
    s += i;  
    i++;  
}
```



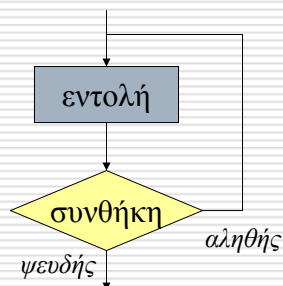
N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Εντολές και έλεγχος ροής (iv)

do
 εντολή
while (συνθήκη);

□ Εντολή do-while

```
int i = 1, s = 0;  
do  
    s += i++;  
while (i <= 10);
```



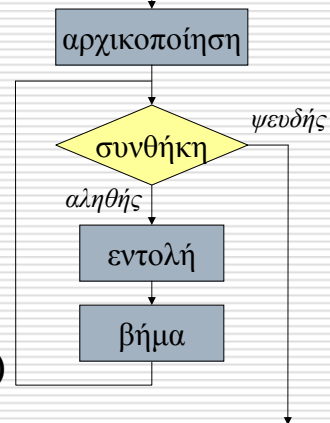
N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Εντολές και έλεγχος ροής (v)

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

□ Εντολή for

```
int i, s;  
for (i=1, s=0; i <= 10; i++)  
    s += i;
```



N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Εντολές και έλεγχος ροής (vi)

□ Εντολή break

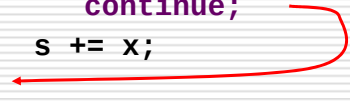
```
int s;  
for (i=0, s=0; i < 10; i++) {  
    int x;  
    scanf("%d", &x);  
    if (x < 0)  
        break;  
    s += x;  
}  
printf("Sum is: %d\n", s);
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Εντολές και έλεγχος ροής (vii)

□ Εντολή continue

```
int s;  
for (i=0, s=0; i < 10; i++) {  
    int x;  
    scanf("%d", &x);  
    if (x < 0)  
        continue;  
    s += x;  
}  
printf("Sum is: %d\n", s);
```



N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Εντολές και έλεγχος ροής (viii)

□ Εντολή switch

```
switch (ch) {  
    case 'a':  
        printf("alpha\n");  
        break;  
    case 'b':  
    case 'c':  
        printf("beta or c\n");  
        break;  
    default:  
        printf("other\n");  
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

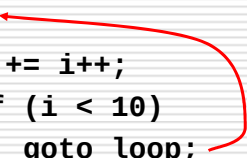
Εντολές και έλεγχος ροής (ix)

□ Ετικέτες και εντολή goto

```
int i = 1, s = 0;

loop:
    s += i++;
    if (i < 10)
        goto loop;

printf("The sum is %d\n", s);
```



□ Όχι goto: δομημένος προγραμματισμός!

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δομή του προγράμματος

□ Το πρόγραμμα αποτελείται από:

- συναρτήσεις
- καθολικές μεταβλητές

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

C και Pascal

□ Διαφορές από την Pascal:

- κύριο πρόγραμμα, διαδικασίες και συναρτήσεις δε διαφοροποιούνται
- το κύριο πρόγραμμα ονομάζεται **main**
- οι διαδικασίες έχουν αποτέλεσμα **void**
- όλες οι συναρτήσεις στο ίδιο επίπεδο
- δεν επιτρέπονται φωλιασμένες συναρτήσεις
- μόνο πέρασμα κατά τιμή (call by value)

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Παράδειγμα

```
int f (int x, int y)
{
    return x * y;
}

int a;

void p (int x)
{
    a = f(x, x+1);
}

void main ()
{
    p(6);
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες (arrays) (i)

- Ακολουθία μεταβλητών
 - με το ίδιο όνομα,
 - με τον ίδιο τύπο δεδομένων,
 - σε συνεχόμενες θέσεις μνήμης.
- Παράδειγμα

```
int a[50];
double d1[5], d2[10];
```
- Η αρίθμηση αρχίζει από το 0 !

```
for (i = 0; i < 50; i++) a[i] = i;
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες (arrays) (ii)

- Αρχικοποίηση πινάκων

```
int a[3] = {6, 7, 42};
char c[] = {'a', 'b', 'c', 'd', 'e'};
```
- Συμβολοσειρές
 - Είναι πίνακες χαρακτήρων

```
char s[6] = "abcde";
```
 - Τερματίζονται με τον κενό χαρακτήρα `'\0'`

```
char s[6] = {'a', 'b', 'c',
              'd', 'e', '\0'};
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες (arrays) (iii)

□ Παράδειγμα

```
int a[3] = {5, 6, 7};
int b[3] = {2, 3, 1};
int i;

for (i = 0; i < 3; i++)
    printf("%d times %d is %d\n",
           a[i], b[i], a[i]*b[i]);
```

```
5 times 2 is 10
6 times 3 is 18
7 times 1 is 7
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Πολυδιάστατοι πίνακες(i)

□ Παράδειγμα

```
int a[3][5];
char c[5][10][4];
```

□ Αρχικοποίηση

```
int i[3][3] = { {1, 0, 0},
                {0, 1, 0},
                {0, 0, 1} };

char s[][10] = {
    "my", "name", "is", "joe"
};
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Πολυδιάστατοι πίνακες(ii)

❑ Πολλαπλασιασμός πινάκων

```
double a[3][4] = ... ,  
       b[4][5] = ... ,  
       c[3][5];  
int i, j, k;  
  
for (i=0; i<3; i++)  
    for (j=0; j<5; j++) {  
        c[i][j] = 0.0;  
        for (k=0; k<4; k++)  
            c[i][j] += a[i][k] * b[k][j];  
    }
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δομές δεδομένων (i)

❑ Αλγόριθμος

***Πεπερασμένο σύνολο εντολών που,
όταν εκτελεστούν, επιτυγχάνουν κάποιο
επιθυμητό αποτέλεσμα***

- Δεδομένα εισόδου και εξόδου
- Κάθε εντολή πρέπει να είναι:
 - ❑ καλά ορισμένη
 - ❑ απλή
- Η εκτέλεση πρέπει να σταματά

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δομές δεδομένων (ii)

■ Πολυπλοκότητα

*Μέτρο εκτίμησης της απόδοσης αλγορίθμων
ως συνάρτηση του μεγέθους του
προβλήματος που επιλύουν*

□ Χρόνος εκτέλεσης

□ Χωρητικότητα μνήμης που απαιτείται

■ Ακριβής μέτρηση πολυπλοκότητας

$$t = f(n)$$

■ Τάξη μεγέθους, συμβολισμοί O , Ω , Θ

$$t = O(f(n))$$

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δομές δεδομένων (iii)

□ Ορισμός των κλάσεων O , Ω , Θ

■ Άνω όριο

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

■ Κάτω όριο

$$g = \Omega(f) \Leftrightarrow \exists c. \exists n_0. \forall n > n_0. g(n) > c f(n)$$

■ Τάξη μεγέθους

$$g = \Theta(f) \Leftrightarrow \exists c_1, c_2. \exists n_0. \forall n > n_0. c_1 f(n) < g(n) < c_2 f(n)$$

□ Διάταξη μερικών κλάσεων πολυπλοκότητας

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) \\ < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δομές δεδομένων (iv)

- Αφαίρεση δεδομένων (data abstraction)
 - Διαχωρισμός **ιδιοτήτων** και **υλοποίησης**
 - **Ιδιότητες** ενός τύπου δεδομένων:
ο τρόπος με τον οποίο δημιουργεί κανείς και χρησιμοποιεί δεδομένα αυτού του τύπου
 - **Υλοποίηση** ενός τύπου δεδομένων:
ο τρόπος με τον οποίο αναπαρίστανται τα δεδομένα στη μνήμη του υπολογιστή και προγραμματίζονται οι διαθέσιμες πράξεις

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δομές δεδομένων (v)

- Οι λεπτομέρειες υλοποίησης επηρεάζουν την **πολυπλοκότητα** των αλγορίθμων
- Είναι συχνή επομένως η **αλλαγή τρόπου υλοποίησης** κάποιου τύπου δεδομένων
- Η **αφαίρεση δεδομένων** ελαχιστοποιεί τις αλλαγές που απαιτούνται στο πρόγραμμα που χρησιμοποιεί έναν τύπο δεδομένων, όταν αλλάξει ο τρόπος υλοποίησης

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Αφηρημένοι τύποι δεδομένων

- Αφηρημένος τύπος δεδομένων – ΑΤΔ (abstract data type)
 - καθορίζει τις **ιδιότητες** του τύπου δεδομένων
 - δεν καθορίζει την **υλοποίησή** του
- Αλγεβρικός ορισμός ΑΤΔ
 - **σύνταξη**: όνομα του τύπου και επικεφαλίδες των πράξεων
 - **σημασιολογία**: κανόνες που περιγράφουν τη λειτουργία των πράξεων

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Παράδειγμα ΑΤΔ: Σύνολα (i)

- Σύνταξη
 - Όνομα τύπου: **set** (σύνολα ακεραίων)
 - Επικεφαλίδες πράξεων:

```
const set empty;  
set add (int x, set s);  
boolean member (int x, set s);  
set union (set s1, set s2);  
boolean subset (set s1, set s2);  
boolean equal (set s1, set s2);
```
- Θεωρούμε ότι έχει οριστεί ο τύπος **boolean**.

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Παράδειγμα ΑΤΔ: Σύνολα (ii)

□ Σημασιολογία

■ Αξιωματικός ορισμός **member**:

`member(x, empty) = false`

`member(x, add(x, s)) = true`

`member(x, add(y, s)) = member(x, s)`

$\vee x \neq y$

■ Αξιωματικός ορισμός **union**:

`union(empty, s) = s`

`union(add(x, s1), s2) =`

`add(x, union(s1, s2))`

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Παράδειγμα ΑΤΔ: Σύνολα (iii)

□ Σημασιολογία

■ Αξιωματικός ορισμός **subset**:

`subset(empty, s) = true`

`subset(add(x, s1), s2) =`

`band(member(x, s2), subset(s1, s2))`

■ Αξιωματικός ορισμός **equal**:

`equal(s1, s2) =`

`band(subset(s1, s2), subset(s2, s1))`

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Συγκεκριμένοι τύποι δεδομένων

- ❑ Συγκεκριμένος τύπος δεδομένων – ΣΤΔ (concrete data type)
 - ❑ καθορίζει τις **ιδιότητες** του τύπου δεδομένων
 - ❑ καθορίζει επακριβώς την **υλοποίησή** του
- ❑ Κάθε γλώσσα προγραμματισμού υποστηρίζει ορισμένους ΣΤΔ, π.χ.
 - ❑ **απλοί τύποι**: ακέραιοι αριθμοί, πραγματικοί αριθμοί, χαρακτήρες, λογικές τιμές
 - ❑ **σύνθετοι τύποι**: πίνακες (arrays), εγγραφές (records), δείκτες (pointers), σύνολα (sets)

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Σχέση ΑΤΔ και ΣΤΔ (i)

- ❑ ΑΤΔ **set**: σύνολα ακεραίων $s = \{ 1, 3, 7, 9 \}$
- ❑ **Υλοποίηση 1**: πίνακας από boolean

false	true	false	true	false	false	false	true	false	true	false
s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]	s[10]

- ❑ **Υλοποίηση 2**: bits ενός πίνακα από int

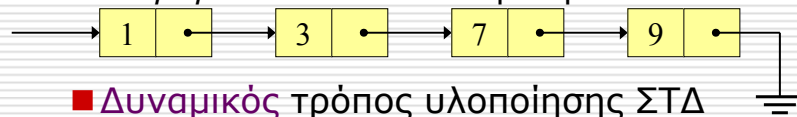
0	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0
s[1]				s[3]				s[7]				s[9]			

- **Στατικός** τρόπος υλοποίησης ΣΤΔ

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Σχέση ΑΤΔ και ΣΤΔ (ii)

- Υλοποίηση 3: απλά συνδεδεμένη λίστα



- Δυναμικός τρόπος υλοποίησης ΣΤΔ

- Υλοποίηση 4: συμβολοσειρά που περιέχει το μαθηματικό ορισμό του συνόλου

"{x | x=1 ∨ x=3 ∨ x=7 ∨ x=9}"

"{x | 1<=x<=10 ∧ x%2=1 ∧ x<>5}"

- Δύσκολη η υλοποίηση των πράξεων

- Απειροσύνολα: "{x | x > 100}"

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες ως ΑΤΔ (i)

- Βασική πράξη: προσπέλαση στοιχείου
a[i]

- Συνήθως υλοποιούνται με κάποιο ΣΤΔ πινάκων (arrays)

- Κόστος προσπέλασης: O(1)

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες ως ΑΤΔ (i)

- Ο ΣΤΔ του μονοδιάστατου πίνακα επαρκεί για την υλοποίηση κάθε ΑΤΔ πίνακα
 - Συνάρτηση *loc* υπολογίζει τη θέση ενός στοιχείου του ΑΤΔ πίνακα στο μονοδιάστατο ΣΤΔ πίνακα της υλοποίησης

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες ως ΑΤΔ (ii)

- ΑΤΔ πίνακα δύο διαστάσεων $n \times m$

$i=1$	0	1	2	3	4	5	
$i=2$	6	7	8	9	10	11	$n=3$
$i=3$	12	13	14	15	16	17	$m=6$
	$j=1$	2	3	4	5	6	

$$loc(n, m, i, j) = m(i - 1) + j - 1$$

- Αρίθμηση κατά στήλες

$$loc(n, m, i, j) = n(j - 1) + i - 1$$

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες ως ΑΤΔ (iii)

- ΑΤΔ κάτω τριγωνικού πίνακα $n \times n$

$i=1$	0				
$i=2$	1	2			
$i=3$	3	4	5		
$i=4$	6	7	8	9	
$i=5$	10	11	12	13	14
	$j=1$	2	3	4	5

$n=5$

$$loc(n, i, j) = i(i-1)/2 + j - 1$$

- Ομοίως για συμμετρικούς πίνακες

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες ως ΑΤΔ (iv)

- ΑΤΔ τριδιαγώνιου πίνακα $n \times n$

$i=1$	0	1			
$i=2$	2	3	4		
$i=3$		5	6	7	
$i=4$			8	9	10
$i=5$				11	12
	$j=1$	2	3	4	5

$n=5$

$$loc(n, i, j) = 2i + j - 3$$

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες ως ΑΤΔ (v)

□ ΑΤΔ αραιού πίνακα $n \times m$

$i = 1$	a_1				
$i = 2$			a_2		
$i = 3$		a_3	a_4		
$i = 4$					a_5
	$j = 1$	2	3	4	5

$n = 4$
 $m = 5$

■ Υλοποίηση με δυαδικό πίνακα

■ Υλοποίηση με τρεις πίνακες

$row = [1, 2, 3, 3, 4]$ $col = [1, 3, 2, 3, 5]$

$val = [a_1, a_2, a_3, a_4, a_5]$

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Αναζήτηση σε πίνακες (i)

□ Σειριακή αναζήτηση

■ Τα στοιχεία διατρέχονται κατά σειρά

■ Κόστος: $O(n)$

12	9	72	22	42	99	14	61
↑	↑	↑	↑	↑			
(1)	(2)	(3)	(4)	(5)			

$n = 8$
 $x = 42$

■ Βελτίωση: στοιχείο “φρουρός” (sentinel)

12	9	72	22	42	99	14	61	7
----	---	----	----	----	----	----	----	---

$n = 8$
 $x = 7$

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Αναζήτηση σε πίνακες (ii)

□ Υλοποίηση σε C

```
int ssearch (int a[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (a[i] == x)
            return i;
    return -1;
}
```

N.Πασισπύρου, Β.Βεσκούκης, ΕΜΠ

Αναζήτηση σε πίνακες (iii)

□ Υλοποίηση σε C με φρουρό

```
int ssearch_s (int a[], int n, int x)
{
    int i;
    a[n] = x;
    for (i = 0; a[i] != x; i++);
    return (i < n) ? i : -1;
}
```

N.Πασισπύρου, Β.Βεσκούκης, ΕΜΠ

Αναζήτηση σε πίνακες (iv)

□ Δυαδική αναζήτηση

- Ο πίνακας πρέπει να είναι **ταξινομημένος**
- Κόστος: $O(\log n)$

3	7	14	22	42	61	72	99
			↑	↑	↑		
			(1)	(3)	(2)		

$n = 8$
 $x = 42$

□ Άλλες μέθοδοι αναζήτησης

- Μικρότερο κόστος $\Rightarrow$ **περισσότερος χώρος**
- Πίνακες **κατακερματισμού** (hash tables)

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση πινάκων (i)

□ Ταξινόμηση επιλογής (selection sort)

■ Ιδέα:

για κάθε i κατ' αύξουσα σειρά
βρες το μικρότερο των στοιχείων μετά το
 $a[i]$
αντιμετάθεσέ το με το $a[i]$

- Κόστος: $O(n^2)$
- Βελτίωση: στοιχείο φρουρός

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση πινάκων (ii)

□ Ταξινόμηση επιλογής, υλοποίηση σε C

```
void ssort (int a[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++) {
        int min = a[i], minj = i;
        for (j = i+1; j < n; j++)
            if (a[j] < min)
                min = a[minj = j];
        a[minj] = a[i];
        a[i] = min;
    }
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση πινάκων (iii)

□ Ταξινόμηση εισαγωγής (insertion sort)

- Χρησιμοποιούμε αυτό τον τρόπο όταν ταξινομούμε τα χαρτιά μιας τράπουλας
- Ιδέα:
για κάθε i από το δεύτερο και κατ' αύξουσα σειρά τοποθέτησε το $a[i]$ στη σωστή του θέση μεταξύ των στοιχείων που είναι πριν από αυτό
- Κόστος: $O(n^2)$
- Βελτιώσεις:
 - στοιχείο φρουρός
 - δυαδική εισαγωγή

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση πινάκων (iv)

- Ταξινόμηση εισαγωγής, υλοποίηση σε C

```
void isort (int a[], int n)
{
    int i, j;
    for (i = 1; i < n; i++) {
        int x = a[i];
        for (j = i-1; j >= 0; j--)
            if (x < a[j])
                a[j+1] = a[j];
            else
                break;
        a[j+1] = x;
    }
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση πινάκων (v)

- Ταξινόμηση φουσαλίδας (bubble sort)

- Ιδέα:

για κάθε i κατ' αύξουσα σειρά
για κάθε $j > i$ κατὰ φθίνουσα σειρά
αν $a[j-1] > a[j]$ αντιμετάθεσε τα $a[j-1]$ και $a[j]$

- Κόστος: $O(n^2)$

- Βελτιώσεις:

- σταματά αν σε ένα πέρασμα δεν γίνει αντιμετάθεση
- σε ποιο σημείο έγινε η τελευταία αντιμετάθεση
- αλλαγή κατεύθυνσης μεταξύ διαδοχικών περασμάτων (shake sort)

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση πινάκων (vi)

- Ταξινόμηση φουσαλίδας, υλοποίηση σε C

```
void bsort (int a[], int n)
{
    int i, j;
    for (i = 0; i < n; i++)
        for (j = n-1; j > i; j--)
            if (a[j-1] > a[j]) {
                int temp = a[j-1];
                a[j-1] = a[j];
                a[j] = temp;
            }
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση πινάκων (vii)

- Ταξινόμηση με διαμέριση (quick sort)

- Ιδέα:

διάλεξε ένα τυχαίο στοιχείο x του πίνακα

διαμέρισε τον πίνακα, μεταφέροντας:

- τα στοιχεία μικρότερα του x στην αρχή

- τα στοιχεία μεγαλύτερα του x στο τέλος

αναδρομικά, ταξινόμηση τα δύο μέρη

- Κόστος: $O(n^2)$ στη χειρότερη περίπτωση
 $O(n \log n)$ κατά μέσο όρο

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση πινάκων (viii)

□ Quick sort, υλοποίηση σε C

```
void qsort (int a[], int n)
{
    qsort_auxil(a, 0, n-1);
}

void qsort_auxil (int a[], int lower,
                  int upper)
{
    if (lower < upper) {
        int x = a[(lower + upper) / 2];
        int i, j;
        for (i = lower, j = upper;
             i <= j; i++, j--) {
            while (a[i] < x) i++;
            while (a[j] > x) j--;
        }
    }
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση πινάκων (ix)

□ Quick sort, υλοποίηση σε C (συνέχεια)

```
        if (i <= j) {
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
    qsort_auxil(a, lower, j);
    qsort_auxil(a, i, upper);
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση πινάκων (x)

□ Άλλοι τρόποι ταξινόμησης

- Ταξινόμηση του Shell (shell sort),
κόστος: $O(n^2)$
- Ταξινόμηση σε σωρό (heap sort),
κόστος: $O(n \log n)$
- Ταξινόμηση με συγχώνευση (merge sort),
κόστος: $O(n \log n)$

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ορισμοί τύπων

□ Συνώνυμα απλών τύπων

```
typedef double real;  
real x, y;
```

□ Συνώνυμα σύνθετων τύπων

```
typedef double vector [10];  
vector v;
```

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

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Απαριθμήσεις (enumerations)

□ Ο τύπος enum

```
enum color_tag {  
    GREEN, RED, BLUE, WHITE, BLACK  
};  
enum color_tag c;  
c = GREEN;
```

```
typedef enum {  
    GREEN, RED, BLUE, WHITE, BLACK  
} color;  
color c = WHITE;
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

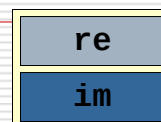
Δομές (structures) (i)

□ Ο τύπος struct

```
struct complex_tag {  
    double re, im;  
};  
struct complex_tag z;  
z.re = 1.0;  
z.im = -1.0;
```

```
typedef struct complex_tag complex;  
complex z;  
complex zs [100];
```

z



N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Δομές (structures) (ii)

□ Παράδειγμα

```
complex complement (complex z)
{
    complex result;

    result.re = z.re;
    result.im = -z.im;
    return result;
}
```

N.Πασισπύρου, Β.Βεσκούκης, ΕΜΠ

Δομές (structures) (iii)

□ Παράδειγμα

```
typedef struct {
    char    firstName [30];
    char    lastName  [50];
    char    phone     [10];
    int     age;
    double  salary;
} record;

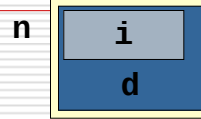
record r [1000];
r[423].age = 32;
```

N.Πασισπύρου, Β.Βεσκούκης, ΕΜΠ

Ενώσεις (unions)

❑ Ο τύπος union

```
union number_tag {  
    int i;  
    double d;  
};  
union number_tag n;  
n.d = 1.2;  
printf("%lf\n", n.d);  
n.i = 42;  
printf("%d\n", n.i);  
printf("%lf\n", n.d);    /* wrong! */
```



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

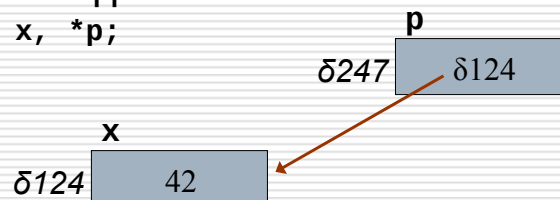
Δείκτες (pointers) (i)

❑ Μεταβλητές

- που δεν περιέχουν μια τιμή κάποιου απλού τύπου δεδομένων
- αλλά τη **διεύθυνση** μιας άλλης μεταβλητής

❑ Παράδειγμα

```
int x, *p;
```



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

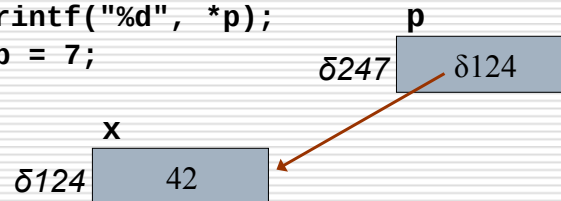
Δείκτες (pointers) (ii)

□ Δεικτοδότηση: &

- η διεύθυνση μιας μεταβλητής
`p = &x;`

□ Αποδεικτοδότηση: *

- το περιεχόμενο μιας διεύθυνσης
`printf("%d", *p);`
`*p = 7;`



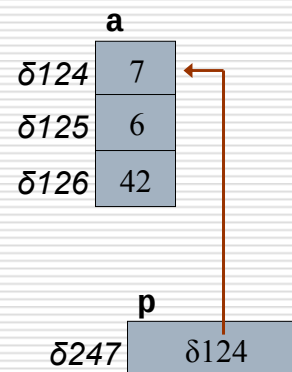
N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες και δείκτες (i)

□ Αριθμητική δεικτών

```
int a[3] = {7, 6, 42};  
int *p;  
p = &a[0];  
p = &a;  
p = a;
```

```
printf("%d\n", *p);  
printf("%d\n", *(p+1));  
p = p+2;  
printf("%d\n", *p);
```



N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

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++; /* correct */
```

```
a++; /* wrong! */
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες και δείκτες (iii)

□ Παράδειγμα: αντιγραφή πινάκων

```
int a[10], b[10];
```

```
int *p = a, *q = b, i;
```

```
/* assume b initialized */
```

```
for (i=0; i<10; i++)
```

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

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Πίνακες και δείκτες (iv)

□ Παράδειγμα: εκτύπωση συμβολοσειράς

```
void putchar (char c);

char s[] = "Hello world!\n";
char *p;

for (p = s; *p != '\0'; p++)
    putchar(*p);
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Κενός δείκτης, δείκτης σε κενό

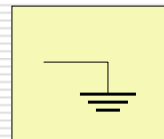
□ Ο κενός δείκτης: **NULL**

- Απαγορεύεται να αποδεικτοδοτηθεί!

```
int *p = NULL;
```

```
*p = 42; /* wrong! */
```

- Ο μόνος δείκτης που αντιστοιχεί στην ψευδή λογική τιμή, όταν χρησιμοποιείται σε συνθήκη



□ Ο δείκτης σε κενό: **void ***

- Γενική μορφή δείκτη
- Απαγορεύεται να αποδεικτοδοτηθεί!
- Απαγορεύεται η αριθμητική δεικτών!

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δείκτες αντί περάσματος με αναφορά

□ Κώδικας Pascal

```
procedure inc (var x : integer);  
begin  
    x := x+1  
end;  
  
... inc(a) ...
```

□ Ισοδύναμος κώδικας C

```
void inc (int * px)  
{  
    (*px)++;  
}  
  
... inc(&a) ...
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δείκτες σε εγγραφές

□ Συντομογραφία

$p \rightarrow x$ είναι ισοδύναμο με $(*p).x$

□ Παράδειγμα

```
struct {  
    int x, y;  
} coordinates, *p;  
  
coordinates.x = 1;  
coordinates.y = 3;  
  
p = &coordinates;  
  
printf("%d\n", p->x);  
printf("%d\n", p->y);
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Μετατροπές τύπων (i)

□ Έμμεσες (coercions)

```
double d = 3;      (με ανάθεση)  
int x = 3.14;
```

```
int f (int x);     (με πέρασμα παραμέτρου)  
f(3.14);
```

□ Άμεσες (type casting)

(τύπος) έκφραση

```
(double) 3  
(int) 3.14  
(int *) NULL
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Μετατροπές τύπων (ii)

□ Πρόβλημα: πώς μπορώ να τυπώσω το αποτέλεσμα της πραγματικής διαίρεσης δυο ακεραίων αριθμών, χωρίς νέες μεταβλητές;

```
int x = 5, y = 3;
```

□ Λάθος λύση #1: `printf("%d", x/y);`

□ Λάθος λύση #2: `printf("%lf", x/y);`

□ Σωστές λύσεις:

⇒ { `printf("%lf", 1.0 * x / y);`
`printf("%lf", (double) x / (double) y);`
`printf("%lf", (double) x / y);`

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δυναμική παραχώρηση μνήμης

□ Συναρτήσεις βιβλιοθήκης <stdlib.h>

■ **void * malloc (size_t n);**

Δυναμική παραχώρηση μνήμης μήκους *n* bytes.
Το αποτέλεσμα πρέπει να μετατραπεί στο σωστό τύπο.
Επιστρέφεται **NULL** αν εξαντληθεί η μνήμη.

■ **void free (void * p);**

Αποδέσμευση της μνήμης στην οποία δείχνει το *p*.
Το *p* πρέπει να έχει δημιουργηθεί με προηγούμενη κλήση στη **malloc**.

□ Πόσα bytes χρειάζονται;

■ **sizeof(type)** π.χ. **sizeof(int)**

■ **sizeof(variable)** π.χ. **sizeof(x)**

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δυναμική παραχώρηση μνήμης

□ Παράδειγμα

```
int *p;  
int i;  
  
p = (int *) malloc(sizeof(int));  
*p = 42;  
free(p);  
  
p = (int *) malloc(10 * sizeof(int));  
for (i = 0; i < 10; i++)  
    p[i] = 42;  
free(p);
```

Το αποτέλεσμα της **malloc** πρέπει να ελεγχθεί ότι δεν είναι **NULL**!

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δυναμικοί ΣΤΔ (i)

- Χρησιμοποιούνται για την υλοποίηση ΑΤΔ
- Παραδείγματα
 - συνδεδεμένες λίστες
 - δέντρα
 - γράφοι
- Πώς υλοποιούνται στη C;
 - με κατάλληλο συνδυασμό **δομών** (**struct**) και **δεικτών** (pointers), και
 - με **δυναμική παραχώρηση** μνήμης

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δυναμικοί ΣΤΔ (ii)

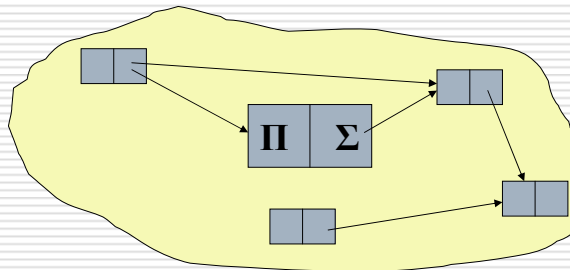
- Πλεονεκτήματα έναντι στατικών ΣΤΔ
 - Δεν επιβάλλουν περιορισμούς στο **μέγιστο πλήθος** των δεδομένων
 - Η μνήμη που χρησιμοποιείται είναι ανάλογη του **πραγματικού πλήθους** των δεδομένων
 - Κάποιες πράξεις υλοποιούνται αποδοτικότερα
- Μειονεκτήματα έναντι στατικών ΣΤΔ
 - Για σταθερό και γνωστό πλήθος δεδομένων, χρησιμοποιούν συνήθως περισσότερη μνήμη
 - Κάποιες πράξεις υλοποιούνται λιγότερο αποδοτικά

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δυναμικοί ΣΤΔ (iii)

□ Ιδέα

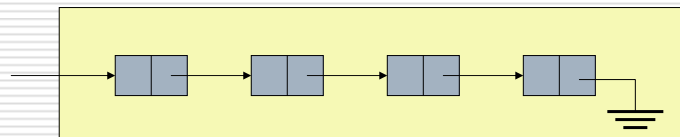
- Κατασκευάζεται ένα σύνολο **κόμβων**
- Κάθε κόμβος περιέχει **πληροφορίες** και **συνδέσμους** προς άλλους κόμβους



Ν.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ

Απλά συνδεδεμένες λίστες (i)

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



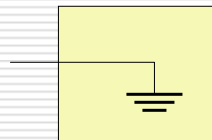
Ν.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ

Απλά συνδεδεμένες λίστες (ii)

□ Παράδειγμα: λίστα ακεραίων

```
struct node_tag {  
    int data;  
    struct node_tag * next;  
};  
  
typedef struct node_tag  
    Node, * LinkedList;
```

□ Κενή λίστα

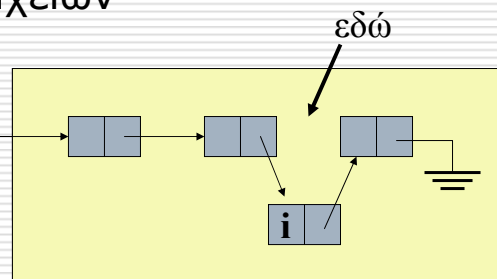


N.Πασισπύρου, Β.Βεσκούκης, ΕΜΠ

Απλά συνδεδεμένες λίστες (iii)

□ Προσθήκη στοιχείων

- απόφαση πού θα προστεθεί
- δημιουργία νέου κόμβου
- αντιγραφή πληροφορίας
- σύνδεση νέου κόμβου

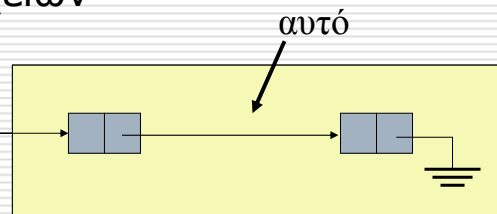


N.Πασισπύρου, Β.Βεσκούκης, ΕΜΠ

Απλά συνδεδεμένες λίστες (iv)

□ Αφαίρεση στοιχείων

- απόφαση ποιο στοιχείο θα αφαιρεθεί
- καταστροφή κόμβου
- σύνδεση υπόλοιπων κόμβων



N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ουρές (i)

□ First In First Out (FIFO)

ό,τι μπαίνει πρώτο, βγαίνει πρώτο

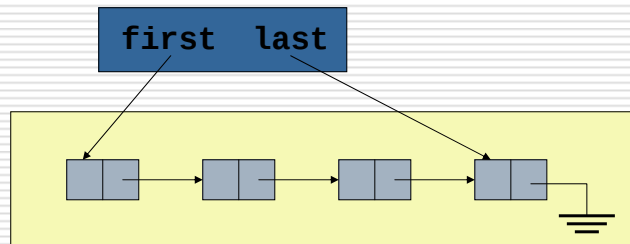
□ Ουρά ακεραίων

- ΑΤΔ: **queue**
- `const queue queueEmpty;`
- `void queueInsert (queue * qp, int t);`
- `int queueRemove (queue * qp);`
- `int queueHead (queue q);`

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ουρές (ii)

- Υλοποίηση με απλά συνδεδεμένη λίστα



Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση ουράς σε C (i)

- Υλοποίηση με απλά συνδεδεμένη λίστα

```
typedef struct list_tag {  
    int data;  
    struct list_tag * next;  
} ListNode;
```

- Τύπος queue

```
typedef struct {  
    ListNode * first;  
    ListNode * last;  
} queue;
```

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση ουράς σε C (ii)

□ Άδεια ουρά

```
const queue queueEmpty = { NULL, NULL };
```

□ Εισαγωγή στοιχείου

```
void queueInsert (queue * qp, int t)
{
    ListNode * n = (ListNode *)
        malloc(sizeof(ListNode));

    if (n == NULL) {
        printf("Out of memory\n");
        exit(1);
    }
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση ουράς σε C (iii)

□ Εισαγωγή στοιχείου (συνέχεια)

```
n->data = t;
n->next = NULL;

if (qp->last == NULL)
    qp->first = qp->last = n;
else {
    qp->last->next = n;
    qp->last = n;
}
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση ουράς (iv)

□ Αφαίρεση στοιχείου

```
int queueRemove (queue * qp)
{
    ListNode * n;
    int result;

    if (qp->first == NULL) {
        printf("Nothing to remove"
               " from an empty queue\n");
        exit(1);
    }
}
```

N.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση ουράς (v)

□ Αφαίρεση στοιχείου (συνέχεια)

```
n = qp->first;
result = qp->first->data;
qp->first = qp->first->next;
free(n);

if (qp->first == NULL)
    qp->last = NULL;

return result;
}
```

N.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση ουράς (vi)

❑ Εξέταση στοιχείου

```
int queueHead (queue q)
{
    if (q.first == NULL) {
        fprintf(stderr, "Nothing to see"
            " in an empty queue\n");
        exit(1);
    }
    return q.first->data;
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Στοιβες(i)

❑ Last In First Out (LIFO)

ό,τι μπαίνει τελευταίο, βγαίνει πρώτο

❑ Στοιβα ακεραιων

■ ΑΤΔ: **stack**

■ `const stack stackEmpty;`

■ `void stackPush (stack * sp, int t);`

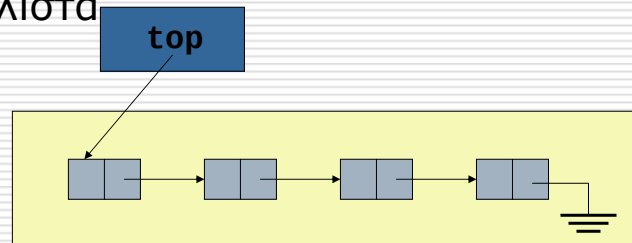
■ `int stackPop (stack * sp);`

■ `int stackTop (stack s);`

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Στοιβες(ii)

- Υλοποίηση με απλά συνδεδεμένη λίστα



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση στοίβας σε C (i)

- Τύπος stack
`typedef ListNode * stack;`
- Άδεια στοίβα
`const stack stackEmpty = NULL;`
- Εισαγωγή στοιχείου

```
void stackPush (stack * sp, int t)
{
    ListNode * n = (ListNode *)
        malloc(sizeof(ListNode));

    if (n == NULL) {
        printf("Out of memory\n");
        exit(1);
    }
```

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση στοίβας σε C (ii)

- Εισαγωγή στοιχείου (συνέχεια)

```
n->data = t;  
n->next = *sp;  
*sp = n;  
}
```

- Αφαίρεση στοιχείου

```
int stackPop (stack * sp)  
{  
    ListNode * n;  
    int result;  
    if (*sp == NULL) {  
        printf("Nothing to remove"  
              " from an empty stack\n");  
        exit(1);  
    }  
}
```

N.Πασισπύρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση στοίβας σε C (iii)

- Αφαίρεση στοιχείου (συνέχεια)

```
n = *sp;  
result = (*sp)->data;  
*sp = (*sp)->next;  
free(n);  
return result;  
}
```

N.Πασισπύρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση στοίβας σε C (iv)

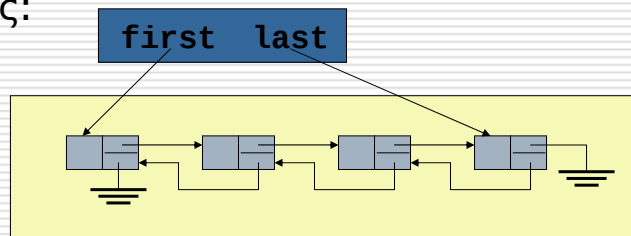
□ Εξέταση στοιχείου

```
int stackTop (stack s)
{
    if (s == NULL) {
        printf("Nothing to see"
              " in an empty stack\n");
        exit(1);
    }
    return s->data;
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διπλά συνδεδεμένες λίστες (i)

- Επίσης γραμμικές διατάξεις
- Δυο σύνδεσμοι σε κάθε κόμβο, προς τον επόμενο και προς τον προηγούμενο
- Γενική μορφή, π.χ. για υλοποίηση ουράς:



N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διπλά συνδεδεμένες λίστες (ii)

- Τύπος κόμβου DListNode

```
typedef struct DListNode_tag {
    int data;
    struct DListNode_tag * next;
    struct DListNode_tag * prev;
} DListNode;
```
- Τύπος dlist

```
typedef struct {
    DListNode * first;
    DListNode * last;
} dlist;
```
- Άδεια λίστα

```
const dlist dlistEmpty = { NULL, NULL };
```

Ν.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ

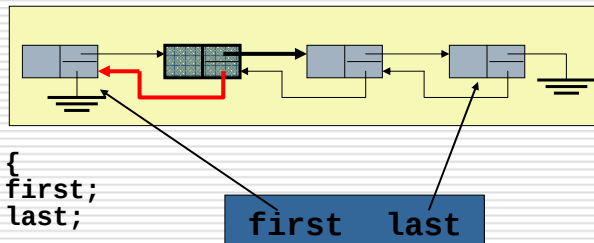
Διπλά συνδεδεμένες λίστες (iii)

- Τύπος κόμβου DListNode

```
typedef struct DListNode_tag {
    int data;
    struct DListNode_tag * next;
    struct DListNode_tag * prev;
} DListNode;
```

- Τύπος dlist

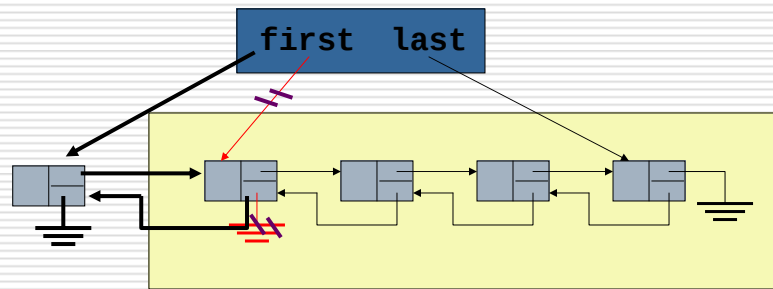
```
typedef struct {
    DListNode * first;
    DListNode * last;
} dlist;
```



Ν.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ

Διπλά συνδεδεμένες λίστες (iv)

- Εισαγωγή στοιχείου στην αρχή



N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Διπλά συνδεδεμένες λίστες (v)

- Εισαγωγή στοιχείου στην αρχή

```
void dlistInsert (dlist * lp, int t)
{
    DListNode * n = (DListNode *)
        malloc(sizeof(DListNode));

    if (n == NULL) {
        fprintf(stderr, "Out of memory\n");
        exit(1);
    }

    n->data = t;
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Διπλά συνδεδεμένες λίστες (vi)

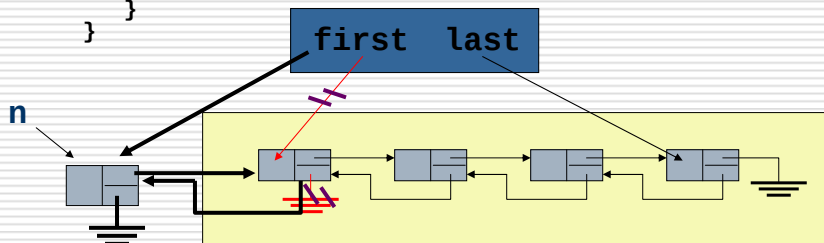
□ Εισαγωγή στοιχείου στην αρχή
(συνέχεια)

```
if (lp->first == NULL) {  
    n->prev = n->next = NULL;  
    lp->first = lp->last = n;  
}  
else {  
    n->prev = NULL;  
    n->next = lp->first;  
    lp->first->prev = n;  
    lp->first = n;  
}  
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

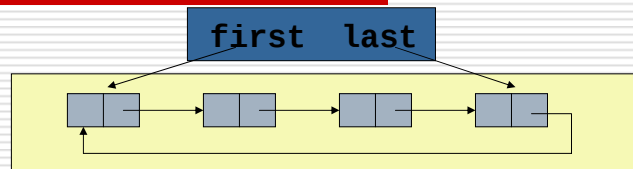
Διπλά συνδεδεμένες λίστες (vii)

```
if (lp->first == NULL) {  
    n->prev = n->next = NULL;  
    lp->first = lp->last = n;  
}  
else {  
    n->prev = NULL;  
    n->next = lp->first;  
    lp->first->prev = n;  
    lp->first = n;  
}  
}
```



N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Κυκλικές λίστες (i)



□ Τύπος clist

```
typedef struct {
    ListNode * first;
    ListNode * last;
} clist;
```

□ Άδεια λίστα

```
const clist clistEmpty = { NULL, NULL };
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Κυκλικές λίστες (ii)

□ Εισαγωγή στοιχείου

```
void clistInsert (clist * lp, int t)
{
    ListNode * n = (ListNode *)
        malloc(sizeof(ListNode));

    if (n == NULL) {
        fprintf(stderr, "Out of memory\n");
        exit(1);
    }

    n->data = t;
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Κυκλικές λίστες (iii)

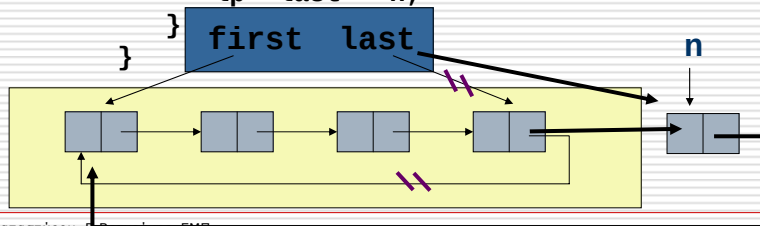
□ Εισαγωγή στοιχείου (συνέχεια)

```
if (lp->first == NULL) {  
    lp->first = lp->last = n;  
    n->next = n;  
}  
else {  
    n->next = lp->first;  
    lp->last->next = n;  
    lp->last = n;  
}  
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Κυκλικές λίστες (iv)

```
if (lp->first == NULL) {  
    lp->first = lp->last = n;  
    n->next = n;  
}  
else {  
    n->next = lp->first;  
    lp->last->next = n;  
    lp->last = n;  
}  
}
```



N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Κυκλικές λίστες (v)

□ Αφαίρεση στοιχείου

```
int clistRemove (clist * lp)
{
    int result;

    if (lp->first == NULL) {
        fprintf(stderr, "Nothing to remove"
            " from empty list\n");
        exit(1);
    }

    result = lp->first->data;
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Κυκλικές λίστες (vi)

□ Αφαίρεση στοιχείου (συνέχεια)

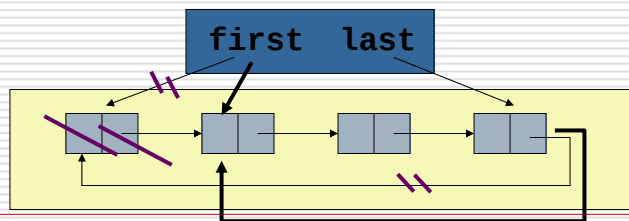
```
    if (lp->first == lp->last) {
        free(lp->first);
        lp->first = lp->last = NULL;
    }
    else {
        lp->first = lp->first->next;
        free(lp->last->next);
        lp->last->next = lp->first;
    }

    return result;
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Κυκλικές λίστες (vii)

```
if (lp->first == lp->last) {
    free(lp->first);
    lp->first = lp->last = NULL;
}
else {
    lp->first = lp->first->next;
    free(lp->last->next);
    lp->last->next = lp->first;
}
return result;
}
```



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Κυκλικές λίστες (viii)

□ Εκτύπωση στοιχείων

```
void clistPrint (clist l)
{
    ListNode * n;
    for (n = l.first; n != NULL;
         n = n->next) {
        printf("%d\n", n->data);
    }
}
```

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Κυκλικές λίστες (ix)

- Εκτύπωση στοιχείων (σωστό!)

```
void clistPrint (clist l)
{
    ListNode * n;
    for (n = l.first; n != NULL;
         n = n->next) {
        printf("%d\n", n->data);
        if (n->next == l.first)
            break;
    }
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ταξινομημένες λίστες (i)

- Τύπος slist

```
typedef ListNode * slist;
```

- Άδεια λίστα

```
const slist slistEmpty = NULL;
```

- Εισαγωγή στοιχείου

```
void slistInsert (slist * lp, int t)
{
    ListNode * n = (ListNode *)
        malloc(sizeof(ListNode));

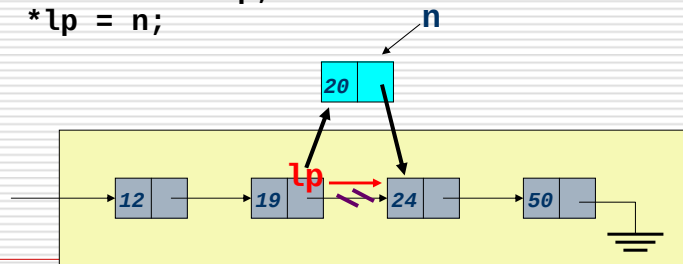
    if (n == NULL) {
        printf("Out of memory\n");
        exit(1);
    }
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ταξινομημένες λίστες (ii)

□ Εισαγωγή στοιχείου (συνέχεια)

```
n->data = t;  
while (*lp != NULL && (*lp)->data < t)  
    lp = &((*lp)->next);  
n->next = *lp;  
*lp = n;  
}
```



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ταξινομημένες λίστες (iii)

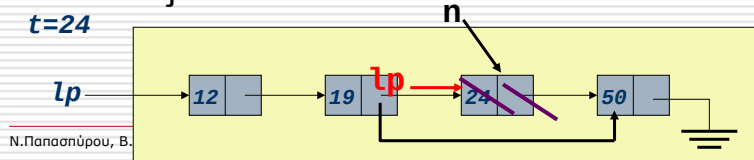
□ Αφαίρεση στοιχείου

```
void slistRemove (slist * lp, int t)  
{  
    ListNode * n;  
    while (*lp != NULL && (*lp)->data < t)  
        lp = &((*lp)->next);  
    if (*lp == NULL) {  
        printf("%d was not found\n", t);  
        exit(1);  
    }  
    n = *lp;  
    *lp = (*lp)->next;  
    free(n);  
}
```

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Ταξινομημένες λίστες (iv)

```
void slistRemove (slist * lp, int t)
{
    ListNode * n;
    while (*lp != NULL && (*lp)->data < t)
        lp = &((*lp)->next);
    if (*lp == NULL) {
        printf("%d was not found\n", t);
        exit(1);
    }
    n = *lp;
    *lp = (*lp)->next;
    free(n);
}
```



Παράμετροι προγράμματος (i)

- ❑ Επικεφαλίδα του προγράμματος
`int main (int argc, char * argv[]);`
- ❑ Παράμετροι
 - `argc` ο αριθμός των παραμέτρων
 - `argv[i]` η i-οστή παράμετρος
 - `argv[0]` το όνομα του προγράμματος
- ❑ Αποτέλεσμα
 - ακέραιος αριθμός που επιστρέφεται στο λειτουργικό σύστημα
 - συνήθως 0 για επιτυχή τερματισμό

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Παράμετροι προγράμματος (ii)

□ Παράδειγμα

```
int main (int argc, char * argv[])
{
    int i;
    printf("Program %s called with "
           "%d parameters:\n", argv[0],
           argc-1);
    for (i = 1; i < argc; i++)
        printf(" %s", argv[i]);
    printf("\nand will return 0\n");
    return 0;
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις εισόδου-εξόδου (i)

□ Βασικές συναρτήσεις εισόδου-εξόδου

```
int printf (const char * format, ...);
int scanf (const char * format, ...);
```

□ Ειδικοί χαρακτήρες στο format

■ Ακέραιοι αριθμοί

%d	στο δεκαδικό σύστημα
%u	χωρίς πρόσημο στο δεκαδικό σύστημα
%o	χωρίς πρόσημο στο οκταδικό σύστημα
%x	χωρίς πρόσημο στο δεκαεξαδικό σύστημα

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις εισόδου-εξόδου (ii)

□ Ειδικοί χαρακτήρες στο format

■ Αριθμοί κινητής υποδιαστολής

%f σε μορφή: [-]ddd.ddd

%e σε μορφή: [-]ddd.ddd e [+/-]ddd

%g σε μορφή %f ή %e

■ Άλλοι τύποι

%c χαρακτήρες

%s συμβολοσειρές

%p δείκτες

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις εισόδου-εξόδου (iii)

□ Παραλλαγές στο format

■ Μέγεθος αριθμών

%h αριθμοί **short** π.χ. %hd, %hx

%l αριθμοί **long** ή **double** π.χ. %ld, %lf

%L αριθμοί **long double** π.χ. %Lf

■ Μήκος αποτελέσματος

%8d αριθμός σε μήκος 8 χαρακτήρων

%20s συμβολοσειρά σε μήκος 20 χαρακτήρων

%+8d αριθμός σε μήκος 8 χαρακτήρων με +

%08d αριθμός σε μήκος 8 χαρακτήρων, τα πρώτα 0

%-8d όπως το %8d με στοίχιση αριστερά

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις εισόδου-εξόδου (iv)

❑ Είσοδος-έξοδος χαρακτήρων

```
int putchar (int c);  
int getchar ();
```

❑ Είσοδος-έξοδος συμβολοσειρών

```
int puts (const char * s);  
char * gets (char * s);
```

❑ Έλεγχος τέλους δεδομένων

```
int eof ();
```

- Η σταθερά **EOF** παριστάνει το τέλος των δεδομένων και έχει τύπο **int**.

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Παράδειγμα

❑ Αντιγραφή δεδομένων

- οι χαρακτήρες που διαβάζονται εκτυπώνονται, μέχρι να παρουσιαστεί τέλος δεδομένων

```
void main ()  
{  
    int c;  
    while ((c = getchar()) != EOF)  
        putchar(c);  
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις διαχείρισης αρχείων (i)

□ Τύπος αρχείου

```
FILE * fp;
```

□ Άνοιγμα αρχείων

```
FILE * fopen (const char * filename,  
             const char * mode);
```

■ Παράμετρος mode

r ανάγνωση (read)

w εγγραφή (write)

a προσθήκη (append)

t κείμενο (text)

b δυαδικά δεδομένα (binary)

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις διαχείρισης αρχείων (ii)

□ Κλείσιμο αρχείων

```
int fclose (FILE * fp);
```

□ Είσοδος-έξοδος χαρακτήρων

```
int fputc (int c, FILE * fp);
```

```
int fgetc (FILE * fp);
```

□ Είσοδος-έξοδος συμβολοσειρών

```
int fputs (const char * s, FILE * fp);
```

```
char * fgets (char * s, int n,  
             FILE * fp);
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις διαχείρισης αρχείων (iii)

- Βασικές συναρτήσεις εισόδου-εξόδου

```
int fprintf (FILE * fp,  
             const char * format, ...);
```

```
int fscanf (FILE * fp,  
            const char * format, ...);
```

- Έλεγχος τέλους αρχείου

```
int feof (FILE * fp);
```

N.Πασασύρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις διαχείρισης αρχείων (iv)

- Είσοδος-έξοδος πολλών δεδομένων

```
size_t fwrite (const void * p,  
               size_t size, size_t num, FILE * fp);
```

```
size_t fread (void * p,  
              size_t size, size_t num, FILE * fp);
```

- Ο ακέραιος τύπος `size_t` χρησιμοποιείται για τη μέτρηση χώρου μνήμης σε bytes.

N.Πασασύρου, Β.Βεσκούκης, ΕΜΠ

Παράδειγμα (i)

□ Αντιγραφή δυαδικών αρχείων

```
int main (int argc, char * argv[])
{
    FILE * fin, * fout;
    unsigned char buffer[1000];
    size_t count;

    fin = fopen(argv[1], "rb");
    if (fin == NULL)
        return 1;

    fout = fopen(argv[2], "wb");
    if (fout == NULL)
        return 2;
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Παράδειγμα (ii)

□ (συνεχίζεται)

```
    while (!feof(fin)) {
        count = fread(buffer, 1,
                        1000, fin);
        fwrite(buffer, 1, count, fout);
    }

    fclose(fin);
    fclose(fout);

    return 0;
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις βιβλιοθήκης (i)

□ Είσοδος και έξοδος <stdio.h>

- Περιέχει όλες τις συναρτήσεις εισόδου-εξόδου
- Προκαθορισμένα αρχεία

FILE * stdin;	τυπική είσοδος
FILE * stdout;	τυπική έξοδος
FILE * stderr;	τυπική έξοδος σφαλμάτων

- Ισοδυναμίες

printf(...) ≡ **fprintf(stdout, ...)**

scanf(...) ≡ **fscanf(stdin, ...)**

κ.λπ.

- Συναρτήσεις διαχείρισης αρχείων με **τυχαία πρόσβαση** (random access)

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις βιβλιοθήκης (ii)

□ Διαχείριση συμβολοσειρών <string.h>

- **size_t strlen (const char * s);**
Μέτρηση αριθμού χαρακτήρων της συμβολοσειράς **s**.
- **char * strcpy (char * s1, const char * s2);**
Αντιγραφή της συμβολοσειράς **s2** στην **s1**.
- **char * strcat (char * s1, const char * s2);**
Προσθήκη της συμβολοσειράς **s2** στο τέλος της **s1**.
- **int strcmp (const char * s1, const char * s2);**
Σύγκριση των συμβολοσειρών **s1** και **s2**.

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Συναρτήσεις βιβλιοθήκης (iii)

❑ Μετατροπή συμβολοσειρών <stdlib.h>

- `int atoi (const char * s);`
Μετατροπή της συμβολοσειράς `s` σε `int`.
- `long int atol (const char * s);`
Μετατροπή της συμβολοσειράς `s` σε `long int`.
- `double atof (const char * s);`
Μετατροπή της συμβολοσειράς `s` σε `double`.

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Παράδειγμα

❑ Μια δυνατή υλοποίηση της `strcmp`

```
int strcmp (const char * s1,
            const char * s2)
{
    while (*s1 == *s2 && *s1 != '\0')
        { s1++; s2++; }

    ≡ while (*s1 && *s1++ == *s2++);

    return (*s1)-(*s2);
}
```

- **Υπόθεση:** οι τιμές του τύπου `char` είναι στο διάστημα 0-255 (όχι αρνητικές)

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Προεπεξεργαστής (preprocessor) (i)

□ Εντολή #include

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

□ Εντολή #define

```
#define MAX_CHARS 1000
char s[MAX_CHARS];

#define INC(x) ((x)++)
INC(a);
INC(*p);
```

N.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ

Προεπεξεργαστής (preprocessor) (ii)

□ Εντολές #ifdef, #ifndef, #else και #endif

```
#define DEBUG

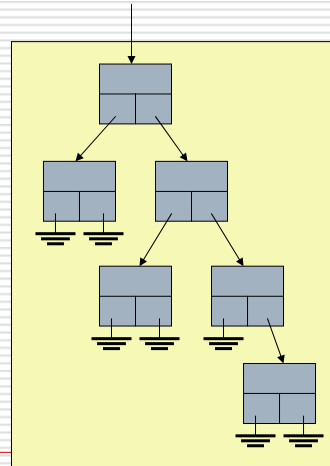
#ifdef DEBUG
    printf("debugging is on\n");
#else
    printf("debugging is off\n")
#endif

#ifndef DEBUG
    printf("optimizations allowed\n");
#endif
```

N.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ

Διαδικτά δένδρα (i)

- ❑ Binary trees
- ❑ Δυο σύνδεσμοι σε κάθε κόμβο
 - αριστερό και δεξιό παιδί
- ❑ Κάθε κόμβος έχει 0, 1 ή 2 παιδιά
- ❑ Ρίζα: ο αρχικός κόμβος του δένδρου
- ❑ Φύλλα: κόμβοι χωρίς παιδιά
- ❑ Βάθος κόμβου: αριθμός συνδέσμων από τη ρίζα



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διαδικτά δένδρα (ii)

- ❑ Τύπος κόμβου `TreeNode`

```
typedef struct TreeNode_tag {
    int data;
    struct TreeNode_tag * left;
    struct TreeNode_tag * right;
} TreeNode;
```
- ❑ Τύπος `tree`

```
typedef TreeNode * tree;
```
- ❑ Κενό δένδρο

```
const tree treeEmpty = NULL;
```

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διαδικά δένδρα (iii)

□ Εισαγωγή σε δένδρα

- Καθοριστική απόφαση: σε ποιο σημείο του δένδρου θα εισαχθεί ο νέος κόμβος
- **Ισοζυγισμένα δένδρα** (balanced trees): το βάθος δυο φύλλων διαφέρει το πολύ κατά 1

□ Συνάρτηση μέγιστου βάθους

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

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διαδικά δένδρα (iv)

□ Εισαγωγή σε ισοζυγισμένα δένδρα

```
void treeBalancedInsert (tree * t, int d)
{
    if (*t == NULL) {
        *t = (TreeNode *)
            malloc(sizeof(TreeNode));

        if (*t == NULL) {
            printf("Out of memory\n");
            exit(1);
        }

        (*t)->data = d;
        (*t)->left = (*t)->right = NULL;
    }
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Δυαδικά δένδρα (v)

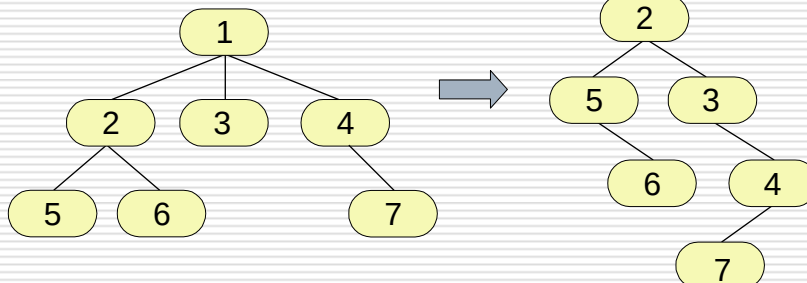
- Εισαγωγή σε ισοζυγισμένα δένδρα (συνέχεια)

```
else {  
    int d1 = treeDepth((*t)->left);  
    int d2 = treeDepth((*t)->right);  
  
    if (d1 <= d2)  
        treeBalancedInsert(  
            &((*t)->left), d);  
    else  
        treeBalancedInsert(  
            &((*t)->right), d);  
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Δένδρα γενικής μορφής

- Κάθε κόμβος k έχει n_k απογόνους, όπου $n_k \in \mathbf{N}$



- Κωδικοποίηση με δυαδικά δένδρα

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διάσχιση δυαδικών δένδρων (i)

□ Σειρά με την οποία διασχίζονται οι κόμβοι

■ κατά **βάθος** (depth first)

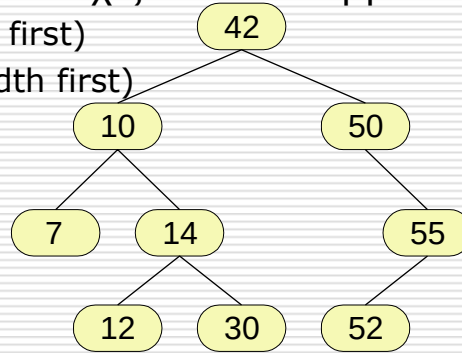
■ κατά **πλάτος** (breadth first)

□ Κατά βάθος

42, 10, 7, 14, 12,
30, 50, 55, 52

□ Κατά πλάτος

42, 10, 50, 7, 14,
55, 12, 30, 52



N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διάσχιση δυαδικών δένδρων (ii)

□ Εκτύπωση κατά βάθος

■ πολύ απλά, με χρήση αναδρομής

□ Υλοποίηση

```
void treePrintDF (tree t)
{
    if (t != NULL) {
        printf("%d ", t->data);
        treePrintDF(t->left);
        treePrintDF(t->right);
    }
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διάσχιση δυαδικών δένδρων (iii)

□ Εκτύπωση κατά πλάτος

- με τη βοήθεια **ουράς** για την αποθήκευση δεικτών προς τους κόμβους που δεν έχουμε επισκεφθεί

□ Υλοποίηση

```
void treePrintBF (tree t)
{
    queue q = queueEmpty;
    if (t != NULL)
        queueInsert(&q, t);
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διάσχιση δυαδικών δένδρων (iv)

□ Εκτύπωση κατά πλάτος, υλοποίηση (συνέχεια)

```
while (!queueIsEmpty(q)) {
    TreeNode * n = queueRemove(&q);

    printf("%d ", n->data);
    if (n->left != NULL)
        queueInsert(&q, n->left);
    if (n->right != NULL)
        queueInsert(&q, n->right);
}
```

N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Αριθμητικές εκφράσεις(i)

□ Παράδειγμα

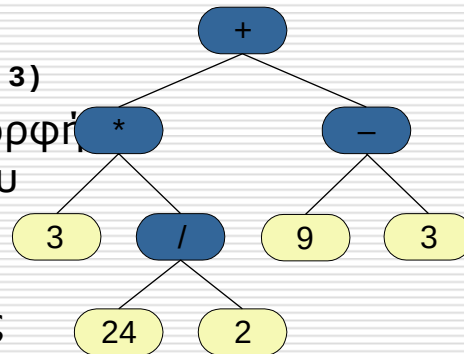
$$3 * (24 / 2) + (9 - 3)$$

□ Παράσταση σε μορφή δυαδικού δένδρου

■ οι αριθμοί
στα φύλλα

■ οι τελεστές στους
υπόλοιπους κόμβους

□ Διάσχιση κατά βάθος



N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Αριθμητικές εκφράσεις(ii)

□ Ενθεματική παράσταση

■ infix notation

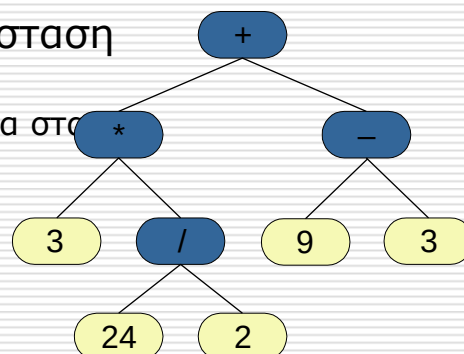
■ ο τελεστής ανάμεσα στα
τελούμενα

■ διφορούμενη:
χρειάζονται
παρενθέσεις

■ η συνήθης μορφή
για τον άνθρωπο

□ Αποτέλεσμα

$$(3 * (24 / 2)) + (9 - 3)$$

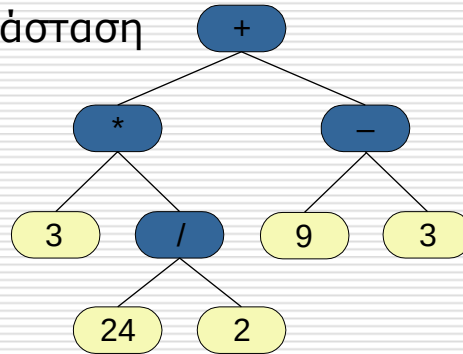


N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Αριθμητικές εκφράσεις(iii)

□ Προθεματική παράσταση

- prefix notation
- ο τελεστής πριν τα τελούμενα
- όχι διφορούμενη, δε χρειάζονται παρενθέσεις
- απλή μηχανική ανάγνωση



□ Αποτέλεσμα

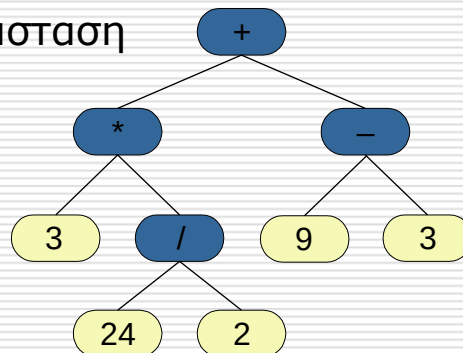
+ * 3 / 24 2 - 9 3

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Αριθμητικές εκφράσεις(iv)

□ Επιθεματική παράσταση

- postfix notation
- ο τελεστής μετά τα τελούμενα
- όχι διφορούμενη, δε χρειάζονται παρενθέσεις
- απλή μηχανική αποτίμηση



□ Αποτέλεσμα

3 24 2 / * 9 3 - +

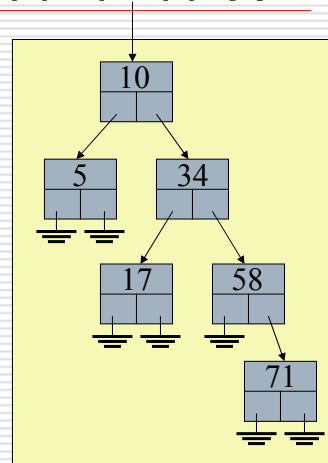
N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δυαδικά δένδρα αναζήτησης (i)

□ Binary search trees

□ Δυαδικά δένδρα με τις παρακάτω ιδιότητες για κάθε κόμβο:

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



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Δυαδικά δένδρα αναζήτησης (ii)

□ Τα δυαδικά δένδρα 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: $O(\log n)$

- υπό την προϋπόθεση το δένδρο να είναι ισοζυγισμένο

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διαδικά δένδρα αναζήτησης(iii)

□ Αναζήτηση

```
TreeNode * treeSearch (tree t, int key)
{
    if (t == NULL)
        return NULL;      /* not found */
    if (t->data == key)
        return t;          /* found */
    if (t->data > key)
        return treeSearch(t->left, key);
    else
        return treeSearch(t->right, key);
}
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δείκτες σε συναρτήσεις

□ Παράδειγμα 1

```
int f (int n);
int (*p) (int n);

p = &f;
printf("%d", (*p)(3));
```

□ Παράδειγμα 2

```
double (*q) (double) = &sqrt;
```

□ Δείκτες ισοδύναμοι με συναρτήσεις

```
p = f;
printf("%d", p(3));
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ολοκλήρωση

```
double integral (double a, double b,  
                 double (*f) (double))  
{  
    const double step = 1e-6;  
  
    double result = 0.0;  
    double x;  
  
    for (x=a; x<=b; x += step)  
        result += f(x) * step;  
    return result;  
}  
  
...  
  
result = integral(0, 3.14159, sin);
```

κακός
αλγόριθμος!

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση φυσαλίδας (i)

□ Τα δεδομένα:

- `void * a` πού βρίσκονται
- `int n` πόσα είναι
- `int size` τί μέγεθος έχει καθένα
- `comparison f` πώς γίνεται η σύγκριση

□ Η σύγκριση των δεδομένων

```
typedef int (*comparison) (void * x,  
                           void * y);
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση φυσαλίδας (ii)

```
void bsort (void * a, int n,
            comparison f, int size)
{
    int i, j;
    for (i = 0; i < n; i++)
        for (j = n-1; j > i; j--) {
            unsigned char * px =
                (unsigned char *) a +
                (j-1) * size;
            unsigned char * py =
                (unsigned char *) a +
                j * size;
```

N.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση φυσαλίδας (iii)

□ (συνέχεια)

```
        if (f(px, py) > 0) {
            int k;
            for (k = 0; k < size; k++) {
                unsigned char temp = *px;

                *px++ = *py;
                *py++ = temp;
            }
        }
    }
}
```

N.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ

Ταξινόμηση φυσαλίδας (iv)

- Συνάρτηση σύγκρισης για ακέραιους

```
int intcompare (void * x, void * y)
{
    return *((int *) x) - *((int *) y);
}
```

- Κλήση για πίνακα ακεραίων

```
int x[] = { 44, 55, 12, 42, ... };
...
bsort(x, n, intcompare, sizeof(int));
```

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

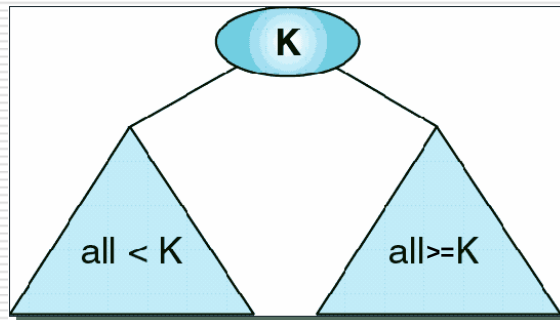
Δέντρα δυαδικής αναζήτησης

- Δενδρικές δομές δεδομένων στις οποίες
 - Όλα τα στοιχεία στο αριστερό υποδέντρο της ρίζας είναι μικρότερα από την ρίζα
 - Όλα τα στοιχεία στο δεξί υποδέντρο της ρίζας είναι μεγαλύτερα ή ίσα με την ρίζα
 - Το αριστερό και το δεξί υποδέντρο είναι δέντρα δυαδικής αναζήτησης
- Binary Search Trees - BSTs

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δέντρα δυαδικής αναζήτησης

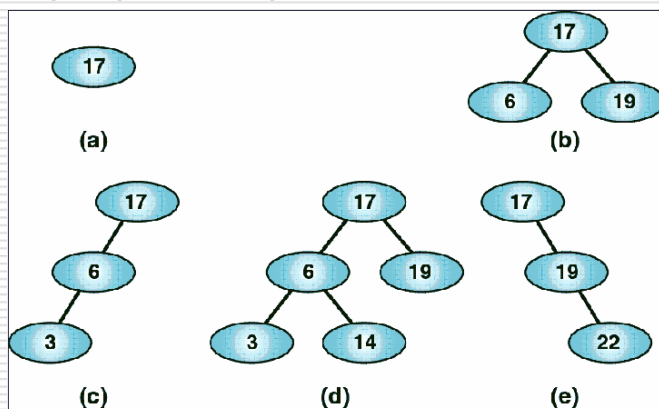
- Η γενική εικόνα ενός τέτοιου δέντρου



Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Παραδείγματα δέντρων BST

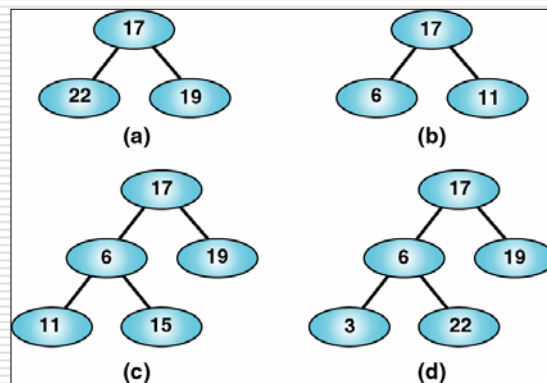
- Εγκυρα δέντρα



Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

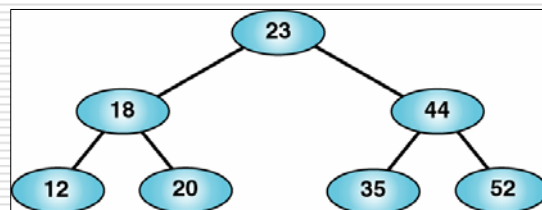
Παραδείγματα δέντρων BST

❑ Μη-έγκυρα δέντρα



Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Διάσχιση δυαδικών δέντρων



- Preorder: 23 18 12 20 44 35 52
- Postorder: 12 20 18 35 52 44 23
- Inorder: 12 18 20 23 35 44 52

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

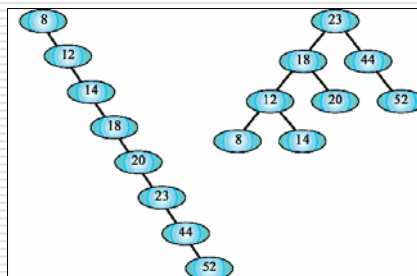
Ζύγισμα δέντρων BST

- Ανάλογα με τη σειρά άφιξης των στοιχείων και τον τρόπο δημιουργίας του δέντρου BST, για τα ίδια στοιχεία δεν προκύπτει πάντα το ίδιο δέντρο
- Το ύψος του δέντρου σχετίζεται με το χρόνο αναζήτησης ενός στοιχείου μέσα στο δέντρο

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ζύγισμα δέντρων BST

- Αναζήτηση με πολυπλοκότητα
 - Από $O(N)$ [χειρότερη περίπτωση]
 - Μέχρι $O(\log_2 N)$ [καλύτερη περίπτωση]



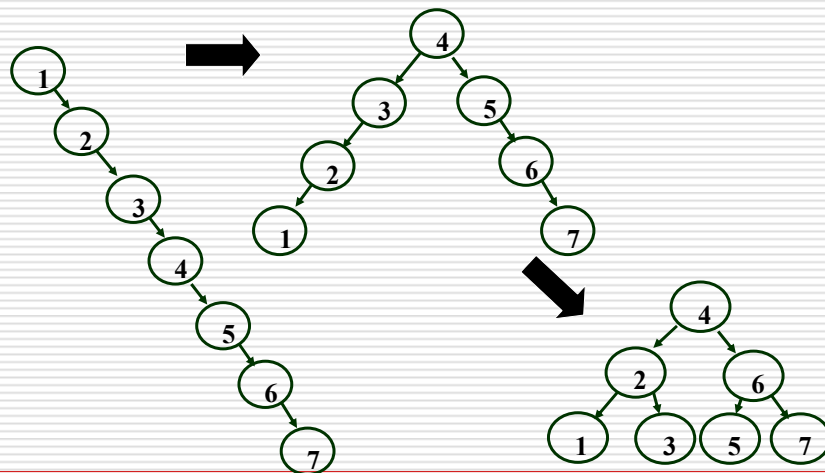
N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ζύγισμα δέντρων BST

- Προκειμένου μια αναζήτηση να διατρέχει όσο το δυνατόν μικρότερο μέρος του δέντρου, αυτό πρέπει να είναι «ζυγισμένο»
 - «Ζυγισμένο»: το βάθος του αριστερού υποδέντρου δεν διαφέρει περισσότερο από 1 από αυτό του δεξιού (ιδανικά: είναι ίσα)
- Αν κατά την προσθήκη νέων στοιχείων στο δέντρο η ζυγαριά «γείρει» δεξιά ή αριστερά, απαιτείται διόρθωση του δέντρου

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Παραδείγματα ζυγισμένων δέντρων



Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Ζυγισμένα δέντρα

- Η περίπτωση όπου το ύψος του αριστερού υποδέντρου είναι **ακριβώς ίσο** με το ύψος του δεξιού, δεν επιτυγχάνεται εύκολα
- Αν από την παραπάνω περίπτωση υπήρχαν «**μικρές**» **αποκλίσεις**, οι επιδόσεις του δέντρου στην αναζήτηση δεν θα επηρεάζονταν ιδιαίτερα

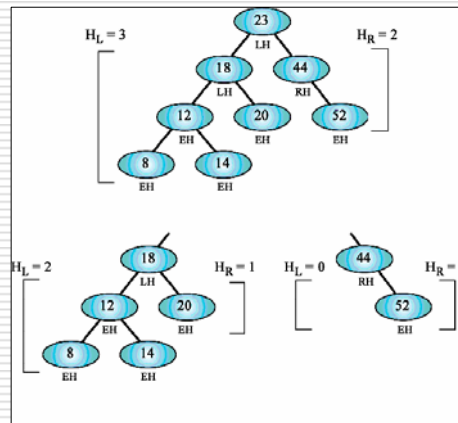
Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δέντρα AVL

- Ένα δέντρο AVL (Adelson-Velskii and Landis) είναι ένα δέντρο BST του οποίου *το ύψος του αριστερού υποδέντρου διαφέρει από αυτό του δεξιού το πολύ κατά 1*
 - Το δεξί και αριστερό υποδέντρο ενός δέντρου AVL είναι επίσης δέντρα AVL
 - Το κενό δέντρο είναι δέντρο AVL
- Οι ιδιότητες αυτές διατηρούνται με παρεμβάσεις (αναδιάρθρωση) καθώς νέα στοιχεία προστίθενται στο δέντρο

Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Δέντρα AVL



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Δέντρα AVL

- Ενα δέντρο AVL λέγεται «ψηλό από αριστερά» όταν το ύψος του αριστερού υποδέντρου είναι μεγαλύτερο από αυτό του δεξιού (κατά πόσο;;;)
- Αντίστοιχα για το «ψηλό από δεξιά»
- Υπάρχουν αρκετοί τρόποι με τους οποίους η προσθήκη ή η διαγραφή στοιχείων σε ένα δέντρο AVL παραβιάζει τη συνθήκη χαρακτηρισμού του

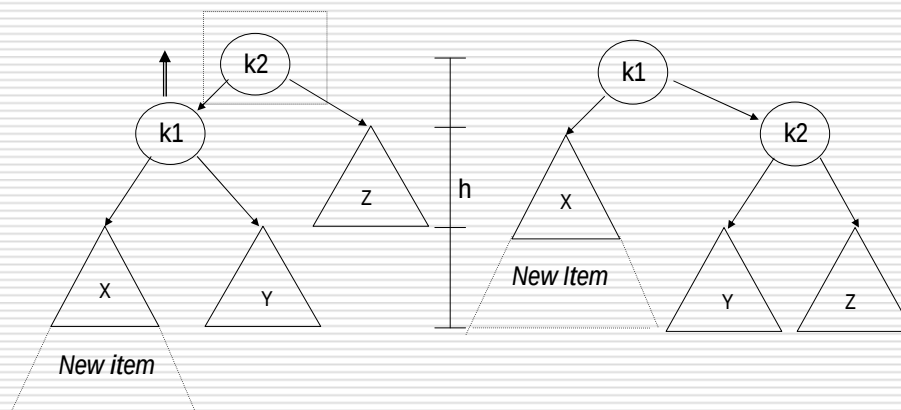
Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Αναδιάταξη δέντρων AVL

- Η χαρακτηριστική ιδιότητα ενός δέντρου AVL μπορεί να παύει να ισχύει με την προσθήκη νέων στοιχείων
- Η επαναφορά της ιδιότητας σε ισχύ, γίνεται με **περιστροφή** του δέντρου, ανάλογα με την περίπτωση αναδιάταξης που συντρέχει

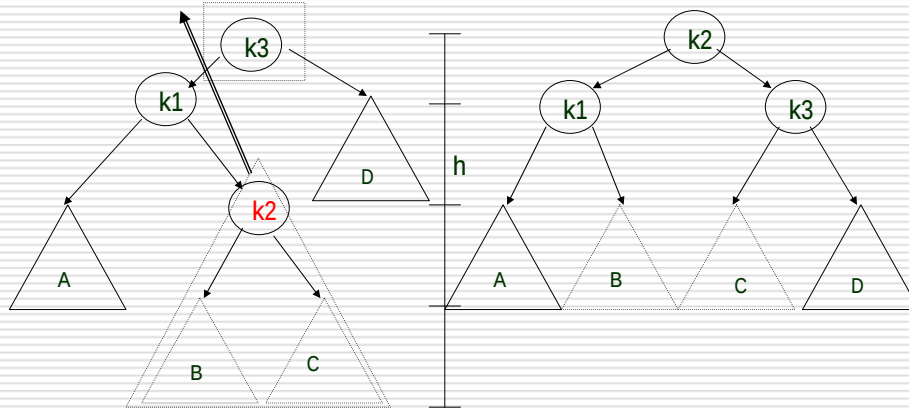
Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Απλή περιστροφή



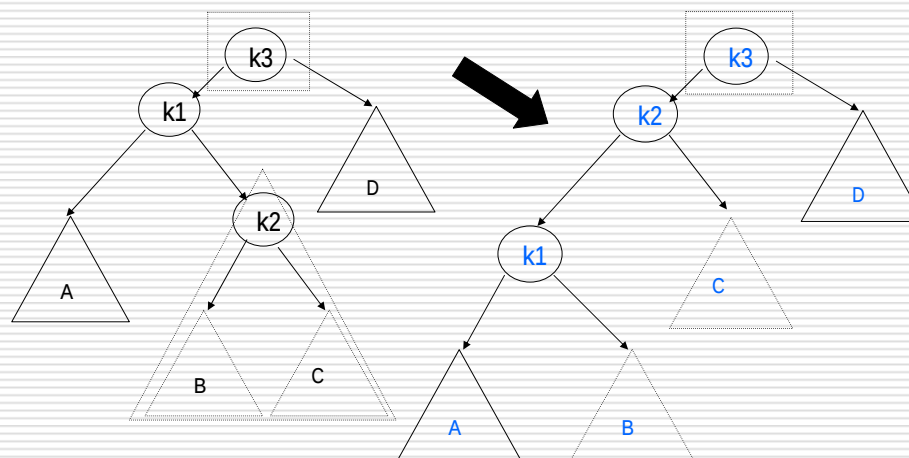
Ν.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Διπλή περιστροφή (α)



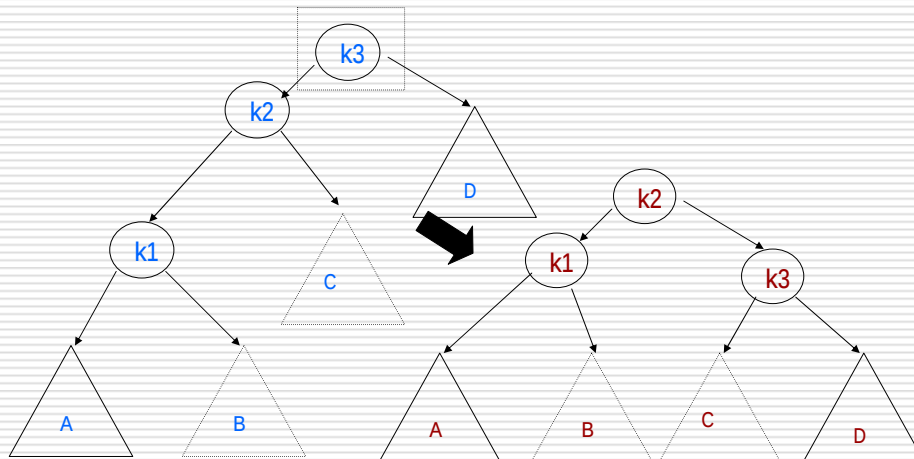
Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διπλή περιστροφή (β)



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Διπλή περιστροφή (γ)



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

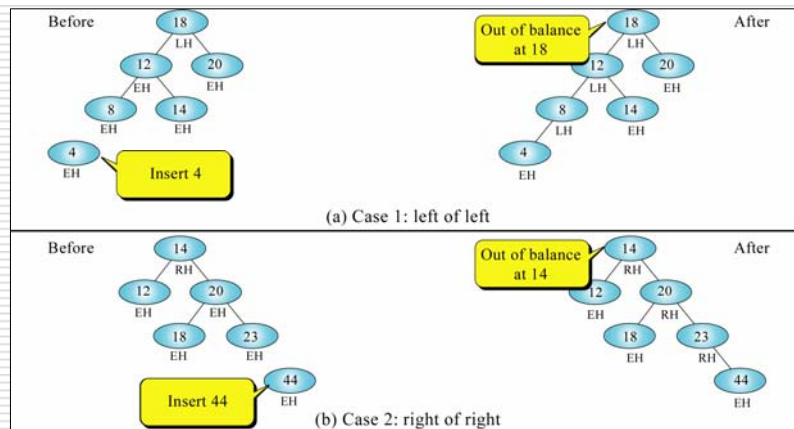
Αναδιάταξη δέντρων AVL

□ Περιπτώσεις αναδιάταξης

- AA: ένα αριστερό υποδέντρο ενός δέντρου AVL που είναι ψηλό από αριστερά, γίνεται επίσης ψηλό από αριστερά (**left of left**)
- ΔΔ: τα αντίστοιχα για το δεξί υποδέντρο (**right of right**)
- ΔΑ: ένα υποδέντρο ενός δέντρου AVL ψηλού από αριστερά, γίνεται ψηλό από δεξιά (**right of left**)
- ΑΔ: ένα υποδέντρο ενός δέντρου AVL ψηλού από δεξιά, γίνεται ψηλό από αριστερά (**left of right**)

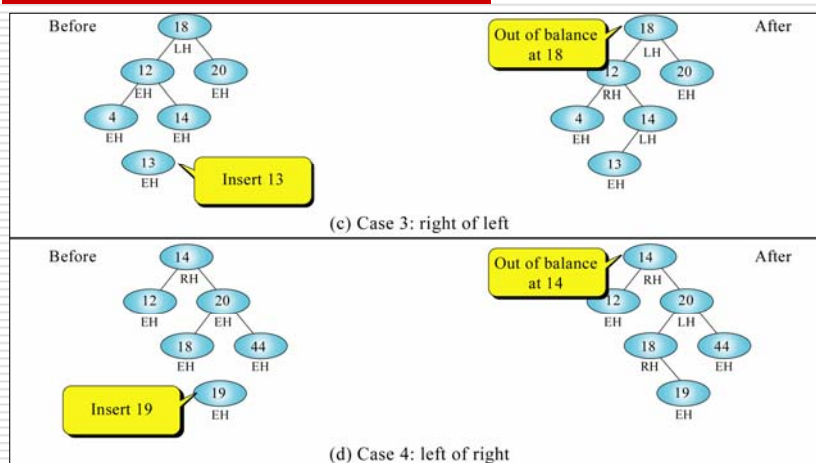
Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Περιπτώσεις αναδιάταξης (α)



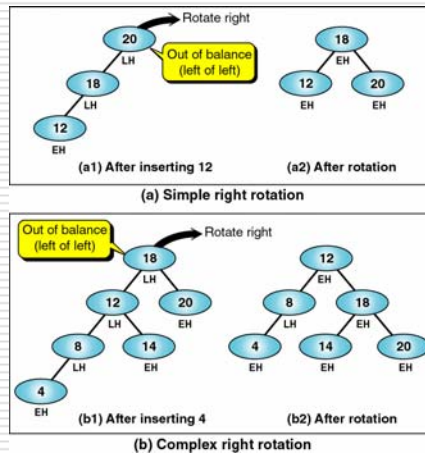
N.Πασσάπου, Β.Βεσκούκης, ΕΜΠ

Περιπτώσεις αναδιάταξης (β)



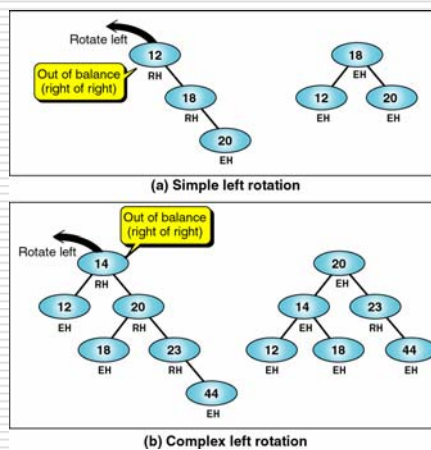
N.Πασσάπου, Β.Βεσκούκης, ΕΜΠ

Αναδιάταξη σε περίπτωση ΑΑ



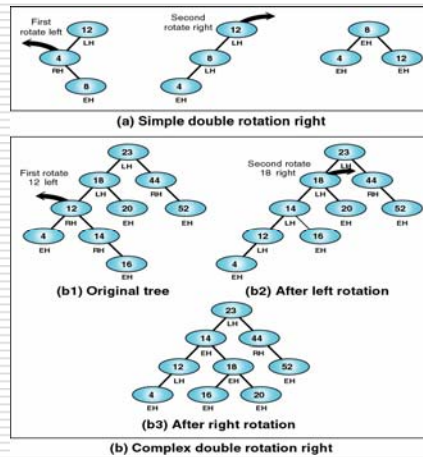
N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Αναδιάταξη σε περίπτωση ΔΔ



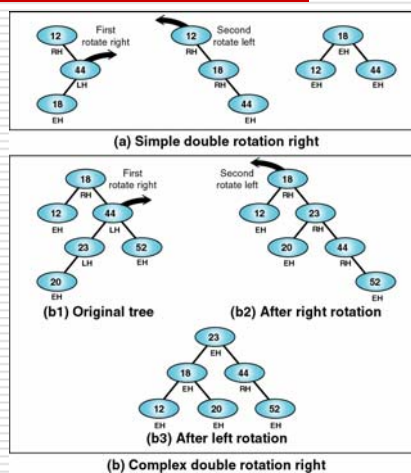
N.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Αναδιάταξη σε περίπτωση ΔΑ



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Αναδιάταξη σε περίπτωση ΑΔ



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

Υλοποίηση δέντρων AVL στη C

□ Μια δομή δεδομένων

Node

key	<keyType>
data	<dataType>
left	<pointer to Node>
right	<pointer to Node>
bal	<LeftH, EvenH, RightH>

End Node

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

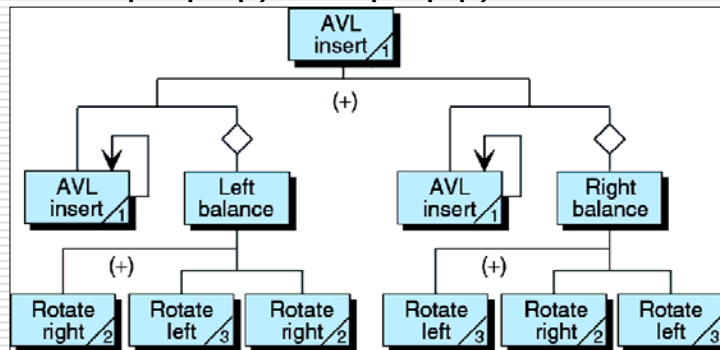
Εισαγωγή σε δέντρα AVL

- Η εισαγωγή νέων στοιχείων γίνεται στα φύλλα, όπως στα δέντρα BST
- Εντοπίζεται το φύλλο στο οποίο θα γίνει η εισαγωγή και δημιουργείται νέος κόμβος
- Γίνεται οπισθοδρόμηση μέχρι την κορυφή του δέντρου, με έλεγχο της ισχύος της συνθήκης ισορροπίας AVL σε κάθε βήμα προς τα πίσω, και επαναφορά σε ισορροπία, όπου απαιτείται

N.Πασιπούρου, Β.Βεσκούκης, ΕΜΠ

Εισαγωγή σε δέντρα AVL

- Διάγραμμα κλήσης μιας αναδρομικής συνάρτησης εισαγωγής



Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

ΑΛΓΟΡΙΘΜΟΣ AVLInsert(ref root <tree pointer>, val newPtr <tree pointer>, ref taller <Boolean>)

```

1 if (root null)
  1 root = newPtr
  2 taller = true
  3 return root
2 end if
3 if (newPtr->key < root->key)
  1 root->left = AVLInsert (root->left, newPtr, taller)
  2 if (taller) // Left subtree is taller
    1 if (root left-high)
      1 root = leftBalance (root, taller)
    2 elseif (root even-high)
      1 root->bal = left-high
    3 else // Was right high -- now even high
      1 root->bal = even-high
      2 taller = false
    4 end if
  4 else // New data >= root data
  
```

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

(Συνέχεια) **AVLInsert(ref root <tree pointer>, val newPtr <tree pointer>, ref taller <Boolean>)**

```
...
4 else // New data >= root data
1 root-> right = AVLInsert (root->right , newPtr, taller)
2 if (taller) // Right subtree is taller
1 if (root left-high)
1 root->bal = even-high
2 taller = false
2 elseif (root even-high) // Was balanced -- now right
high
1 root->bal = right-high
3 else
1 root = rightBalance (root, taller)
4 end if
3 end if
5 end if
6 return root
```

end AVLInsert

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

ΑΛΓΟΡΙΘΜΟΣ

leftBalance (ref root <tree pointer>, ref taller <Boolean>)

```
1 leftTree = root-> left
2 if (leftTree left-high)
// Case 1: Left of left. Single rotation right.
1 rotateRight (root)
2 root->bal = even-high
3 leftTree->bal = even-high
4 taller = false
3 else
// Case 2: Right of left. Double rotation required.
1 rightTree = leftTree->right
// adjust balance factors
2 if (rightTree->bal left-high)
1 root->bal = right-high
2 leftTree->bal = even-high
3 elseif (rightTree->bal = even-high)
1 leftTree->bal = even-high
4 else
```

Ν.Παπασπύρου, Β.Βεσκούκης, ΕΜΠ

(Συνέχεια)

**leftBalance (ref root <tree pointer>, ref taller
<Boolean>)**

```
4 else
  // rightTree->bal is right-high
  1 root->bal = even-high
  2 leftTree->bal = left-high
5 end if
6 rightTree->bal = even-high
7 root->left = rotateLeft (leftTree)
8 root = rotateRight (root)
9 taller = false
4 end if
5 return root

end leftBalance
```

N.Παπασιώρου, Β.Βεσκούκης, ΕΜΠ