

Τύποι Δεδομένων

Ορισμός

Τύπος Δεδομένων (Data Type) είναι *ένα σύνολο ομοειδών δεδομένων* (δηλαδή δεδομένων που έχουν ορισμένα κοινά χαρακτηριστικά), *μαζί με όλες τις πράξεις που ορίζονται και έχουν νόημα πάνω σ' αυτά.*

Παράδειγμα

Ο τύπος δεδομένων των *Ακεραίων (Integer)*, δηλαδή οι αριθμοί ... -3, -2, -1, 0, 1, 2, 3 ... μαζί με τις *πράξεις* που ορίζονται πάνω σ' αυτούς (*πρόσθεση, αφαίρεση, πολλαπλασιασμός, ακέραιο πηλίκο, ακέραιο υπόλοιπο, ύψωση σε δύναμη....*)

Κατηγορίες

- *Απλοί* (στοιχειώδεις) τύποι δεδομένων (**Simple/elementary data types**)
- *Σύνθετοι* ή *Δομημένοι* Τύποι δεδομένων (**Structured Data Types**)

Απλοί (στοιχειώδεις) τύποι δεδομένων

- *Ακέραιοι* (Integer)
- *Πραγματικοί* (Real)
- *Μιγαδικοί* (Complex)
- *Χαρακτήρες* (Character)
- *Λογικοί* (Logical ή Boolean)
- ...

Σύνθετοι ή Δομημένοι Τύποι δεδομένων

- *Διανύσματα / Πίνακες* (Arrays / Tables)
- *Συμβολοσειρές* (Strings)
- *Εγγραφές* (Records)
- *Λίστες* (Lists)
- *Αρχεία* (Files)
- *Σύνολα* (Sets)
- *Λεξικά* (dictionaries)
-

Ακέραιοι (Integer)

- Ο πιο γνωστός και συνηθισμένος τύπος δεδομένων
- Περιλαμβάνει ένα *πεπερασμένο υποσύνολο* των μαθηματικών ακεραίων.
αναμενόμενο, αφού ο ηλεκτρονικός υπολογιστής είναι ένα μηχάνημα με πεπερασμένες υπολογιστικές δυνατότητες και πεπερασμένη μνήμη.
- Στην βασική του εκδοχή ένας άκεραιο αποθηκεύεται σε *2 bytes (16 bits)* μνήμης.
- Αυτό σημαίνει ότι ένας άκεραιο μπορεί να πάρει το πολύ $2^{16} = 65.536$ διαφορετικές τιμές, δηλαδή από το *-32.768* μέχρι και το *32.767*
- Για μεγαλύτερη γκάμα τιμών οι περισσότερες γλώσσες προγραμματισμού διαθέτουν διαφορετικά μεγέθη / παραλλαγές του τύπου των ακεραίων πχ *Long integers* (4 ή 8 bytes) *Short integers* (1 byte)
- Σε ορισμένες γλώσσες προγραμματισμού (πχ Python), το μέγεθος της γκάμας των ακεραίων *περιορίζεται μόνο από την ποσότητα της διαθέσιμης μνήμης*

Τρόπος αποθήκευσης ακεραίων αριθμών στη μνήμη

Οι ακέραιοι αποθηκεύονται σε διαδοχικά bytes στη μνήμη ενός Η/Υ

MSB (most significant byte)

(least significant byte) LSB

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

Υπάρχουν δυο τρόποι (σχήματα) αποθήκευσης

1. Το Most Significant Byte αποθηκεύεται πρώτο (στη χαμηλότερη θέση μνήμης)

ονομάζεται σχήμα αποθήκευσης **Big Endian**
(the "big end" goes first)

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

2. Το Least Significant Byte αποθηκεύεται πρώτο (στη χαμηλότερη θέση μνήμης)

ονομάζεται σχήμα αποθήκευσης **Little Endian**
(the "little end" goes first)

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

Πράξεις πάνω στους ακεραίους

- πρόσθεση $\langle \text{ακέραιος} \rangle + \langle \text{ακέραιος} \rangle = \langle \text{ακέραιος} \rangle$
- αφαίρεση $\langle \text{ακέραιος} \rangle - \langle \text{ακέραιος} \rangle = \langle \text{ακέραιος} \rangle$
- πολλαπλασιασμός $\langle \text{ακέραιος} \rangle \times \langle \text{ακέραιος} \rangle = \langle \text{ακέραιος} \rangle$
- ακέραιο πηλίκο $\langle \text{διαιρετέος} \rangle // \langle \text{διαιρέτης} \rangle = \langle \text{ακέραιο υπόλοιπο} \rangle$
- ακέραιο υπόλοιπο $\langle \text{διαιρετέος} \rangle \% \langle \text{διαιρέτης} \rangle = \langle \text{ακέραιο υπόλοιπο} \rangle$
- ύψωση σε δύναμη

Πραγματικοί (Real)

Όπως και με τους ακέραιους, ο τύπος Πραγματικοί (Real) περιλαμβάνει μόνο ένα πεπερασμένο υποσύνολο του συνόλου των πραγματικών αριθμών της άλγεβρας.

Επιστημονική Αναπαράσταση των πραγματικών αριθμών (Scientific Notation)

- Ειδική μορφή αναπαράστασης αριθμών που χρησιμοποιείται κυρίως σε επιστημονικές μετρήσεις.
- κάθε αριθμός αναπαρίσταται σαν ένα γινόμενο ενός *δεκαδικού αριθμού* επί κάποια *δύναμη του 10*.
- Ανάλογα με την επιλογή της δύναμης του 10, μπορούμε να έχουμε πολλές διαφορετικές επιστημονικές αναπαραστάσεις του ίδιου αριθμού.

Επιστημονική Αναπαράσταση των πραγματικών αριθμών

Παράδειγμα 1: εναλλακτικές μορφές αναπαράστασης του αριθμού **1500**

15×10^2 / 1.5×10^3 / 1500.0×10^0 / 0.15×10^4 / 0.0015×10^6 / 15000×10^{-1}

Επειδή η βάση της δύναμης είναι πάντα το 10, μπορούμε να παραλείψουμε το 10 και να **παριστάνουμε τους αριθμούς μας** μόνο με **το δεκαδικό τους μέρος** και τον εκθέτη της δύναμης.

Για να ξεχωρίσουμε μάλιστα τον εκθέτη, βάζουμε μπροστά του το γράμμα 'E'. Σύμφωνα τώρα με αυτά που είπαμε, οι παραπάνω αριθμοί γράφονται:

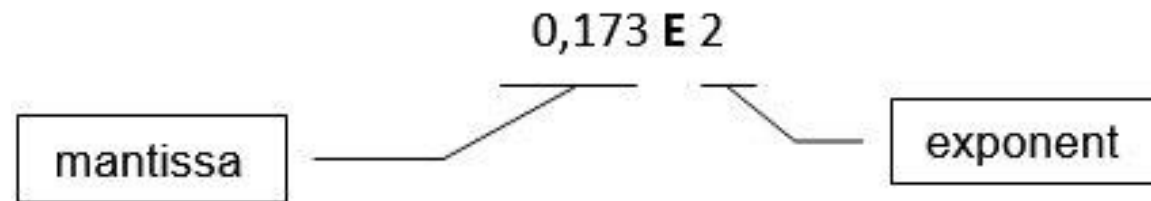
$15E2$ / $1.5E3$ / $1500E0$ / $0.15E4$ / $0.0015E6$ / $15000E-1$

Επιστημονική Αναπαράσταση των πραγματικών αριθμών

Παράδειγμα 2: Μερικές πιθανές επιστημονικές αναπαραστάσεις του αριθμού 17.3

17.3E0 1.73E1 0.173E2 0.000173E5 173.0E-1 1730.0E-2 173000.0E-4

Το πρώτο μέρος (δεκαδικό μέρος) της επιστημονικής αναπαράστασης ενός αριθμού ονομάζεται *mantissa* (δεν υπάρχει αντίστοιχος ελληνικός όρος) και το δεύτερο μέρος (το μέρος εκείνο δηλαδή που ακολουθεί το σύμβολο 'E') ονομάζεται *exponent* (εκθέτης).



Θεωρητικά υπάρχουν *άπειρες* μορφές επιστημονικής αναπαράστασης ενός αριθμού

Αυτός είναι και ο λόγος για τον οποίο οι πραγματικοί αριθμοί ονομάζονται και αλλιώς *αριθμοί κινητής υποδιαστολής* (Floating Point numbers).

Κανονικοποιημένη μορφή επιστημονικής αναπαράστασης

(Normalized Scientific Notation)

Από όλες τις -άπειρες- πιθανές μορφές επιστημονικής αναπαράστασης ενός αριθμού, υπάρχει μια μόνο μορφή στην οποία η mantissa

- έχει ακέραιο μέρος μηδέν
- το πρώτο ψηφίο μετά την υποδιαστολή είναι μη μηδενικό

Η μορφή αυτή ονομάζεται **κανονικοποιημένη** μορφή της επιστημονικής αναπαράστασης.

Πχ η κανονικοποιημένη επιστημονική αναπαράσταση των αριθμών 1500 και 17,3 που είδαμε στα δυο προηγούμενα παραδείγματα είναι αντίστοιχα $0,15\text{E}4$ και $0,173\text{E}2$

Τρόπος αποθήκευσης πραγματικών αριθμών στη μνήμη

Γίνεται μετατρέποντάς έναν αριθμό στην κανονικοποιημένη μορφή της επιστημονικής του αναπαράστασης και εν συνεχεία αποθηκεύοντας ξεχωριστά ένα ζεύγος ακεραίων $[m, e]$

- Ο πρώτος ακέραιος αποτελείται από τα ψηφία που ακολουθούν την υποδιαστολή στην -κανονικοποιημένη- mantissa, και έχει το πρόσημό της *(αποθηκεύεται συνήθως σε 3 bytes)*
- Ο δεύτερος ακέραιος περιέχει τον εκθέτη *(αποθηκεύεται συνήθως σε 1 byte)*

Συνεχίζοντας τα δυο προηγούμενα παραδείγματα,

- ο αριθμός **1500** θα αποθηκευτεί σαν το ζεύγος των ακεραίων **[15 4]**
- ο αριθμός **17,3** θα αποθηκευτεί σαν το ζεύγος των ακεραίων **[173 2]**

Τρόπος αποθήκευσης πραγματικών αριθμών στη μνήμη

Μερικά ακόμα παραδείγματα

| Αριθμός | Διάφορες μορφές επιστημονικής αναπαράστασης | | | Κανονικο ποιημενη | Ζεύγος Ακεραίων |
|---------|--|-------------|-------------|----------------------|--------------------|
| 3,14159 | 3,14159E0 | 314,1590E-2 | 0,0314159E2 | 0,314159E1 | 314159 1 |
| -2000 | -2,0E3 | -0,2E4 | -20000.0E-1 | -0,2E4 | -2 4 |
| 0,0001 | 1.0E-4 | 0,01E-2 | 1000,0E-7 | 0,1E-3 | 1 -3 |

Αντίστροφο παράδειγμα.

Ένας αριθμός τύπου Real είναι αποθηκευμένος ως το ζεύγος ακεραίων $[-25 \quad 1]$.
Επομένως η κανονικοποιημένη του μορφή είναι $-0,25E2$, ή αλλιώς $-0,25 \times 10^1$
δηλαδή πρόκειται για τον αριθμό $-2,5$

Πράξεις πάνω στους πραγματικούς

- πρόσθεση $\langle \text{πραγματικός} \rangle + \langle \text{πραγματικός} \rangle = \langle \text{πραγματικός} \rangle$
- αφαίρεση $\langle \text{πραγματικός} \rangle - \langle \text{πραγματικός} \rangle = \langle \text{πραγματικός} \rangle$
- πολλαπλασιασμός $\langle \text{πραγματικός} \rangle \times \langle \text{πραγματικός} \rangle = \langle \text{πραγματικός} \rangle$
- διαίρεση $\langle \text{πραγματικός} \rangle / \langle \text{πραγματικός} \rangle = \langle \text{πραγματικός} \rangle$
- ύψωση σε δύναμη

Η πρόσθεση, αφαίρεση, πολλαπλασιασμός και διαίρεση *πραγματικών* υλοποιούνται στον ηλεκτρονικό υπολογιστή *με εντελώς διαφορετικές* (και αρκετά περισσότερο περίπλοκες) *διαδικασίες* από τις αντίστοιχες πράξεις των *ακεραίων*

Περιοχή και ακρίβεια πραγματικού αριθμού

← 1 byte →

| |
|----------|
| mantissa |
| mantissa |
| mantissa |
| exponent |

- Η περιοχή (range) των πραγματικών αριθμών που αναγνωρίζονται από τον Η/Υ εξαρτάται από την περιοχή τιμών που μπορεί να πάρει ο *ακέραιος εκθέτης*.
- Συνήθως διατίθεται *1 byte* (στην πραγματικότητα *7 bits*) για την αποθήκευση του εκθέτη, πράγμα που σημαίνει ότι ο εκθέτης μπορεί να παίρνει τιμές από *-127* μέχρι και *127* ($2^7=128$ διαφορετικές τιμές)
- Αυτό πρακτικά σημαίνει ότι μπορούμε να έχουμε πραγματικούς αριθμούς (θετικούς και αρνητικούς) *τόσο μεγάλους όσο περίπου και το 10^{127}* και *τόσο κοντά στο μηδέν όσο περίπου και το 10^{-127}*
- Η ακριβής αναπαράσταση τόσο μεγάλων (ή αντίστοιχα μικρών) αριθμών θα χρειαζόταν περίπου 127 δεκαδικά ψηφία.
- Αντ' αυτού περιοριζόμαστε στα *στο πλήθος των δεκαδικών ψηφίων που μας παρέχει η mantissa*, η οποία εν τέλει και καθορίζει την *ακρίβεια* του πραγματικού αριθμού.

Περιοχή και ακρίβεια πραγματικού αριθμού

Εφαρμογή: Αποθήκευση πραγματικού αριθμού σε 4 bytes / 32 bits

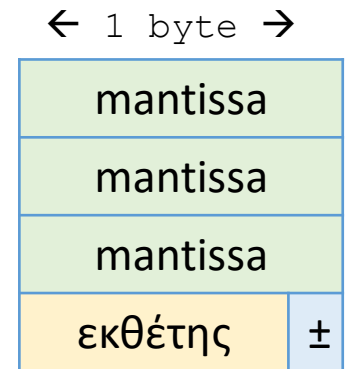
mantissa 24 bits / εκθέτης 7 bits / πρόσημο 1 bit

Περιοχή (range):

$2^7=127$ **[-127...0...127]** επομένως range από -10^{127} έως 10^{127}

Ακρίβεια (precision):

$2^{24} = 16777216$ **1 6 7 7 7 2 1 6** _{1 2 3 4 5 6 7 8} (6-7 δεκαδικά ψηφία [και όχι 8 γιατί])



Παράδειγμα 1

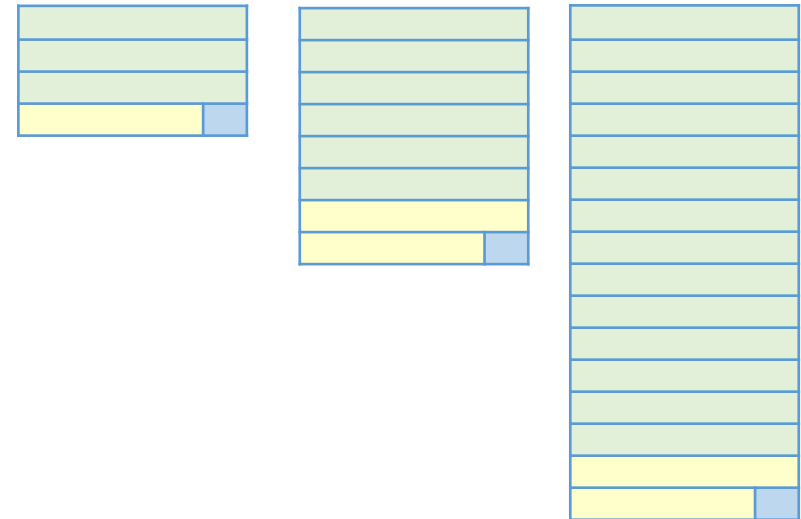
3.14159265 \Rightarrow 0.314159265 **E-1** \Rightarrow [**3141592**65, -1] \Rightarrow [**3141593**, -1] \Rightarrow 3.141593

Παράδειγμα 2

123456789 \Rightarrow 0.123456789 **E9** \Rightarrow [**1234567**89, 9] \Rightarrow [**1234568**, 9] \Rightarrow 123456800

Για ακόμη μεγαλύτερη ακρίβεια...

- *Single Precision* (4 bytes / 32 bits)
 - mantissa 24 bits
 - εκθέτης 7 bits
 - πρόσημο 1 bit
 - παρεχόμενη ακρίβεια *6-7 σημαντικά ψηφία*
- *Double Precision* (8 bytes / 64 bits)
 - mantissa 48 bits
 - εκθέτης 15 bits
 - πρόσημο 1 bit
 - παρεχόμενη ακρίβεια *10-12 σημαντικά ψηφία*
- *Quadruple Precision* (16 bytes / 128 bits)
 - mantissa 112 bits
 - εκθέτης 15 bits
 - πρόσημο 1 bit
 - παρεχόμενη ακρίβεια *33-36 σημαντικά ψηφία*



Χαρακτήρες

- Περιλαμβάνει το σύνολο όλων των συμβόλων που μπορεί να αναγνωρίσει (και κατά συνέπεια να επεξεργαστεί) ένας ηλεκτρονικός υπολογιστής.
 - Στο σύνολο αυτό ανήκουν όλα τα γράμματα της Αγγλικής αλφαβήτου ('A', 'a', 'B', 'b', 'C', 'c' ...), τα ψηφία '0', '1', '2' .. '9', καθώς και ένα πλήθος από ειδικά σύμβολα όπως '+', '-', '=', '/', '?', '!', '\$', '%' κ.λ.π.
 - Υπάρχουν *μόνο δύο πράξεις* που ορίζονται στο σύνολο των χαρακτήρων και μας επιστρέφουν σαν αποτέλεσμα πάλι κάποιον χαρακτήρα (κλειστές πράξεις).
 - **prev()** : Προηγούμενος χαρακτήρας. Επιστρέφει τον χαρακτήρα που βρίσκεται στην *προηγούμενη* θέση σύμφωνα με το ισχύον σύστημα κωδικοποίησης
 - **next()** : Επόμενος χαρακτήρας. Επιστρέφει τον χαρακτήρα που βρίσκεται στην *επόμενη* θέση σύμφωνα με το ισχύον σύστημα κωδικοποίησης
- π.χ. για την κωδικοποίηση σύμφωνα με το πρότυπο *ASCII*, το αποτέλεσμα της πράξης **prev('D')** είναι ο χαρακτήρας 'C' ενώ το αποτέλεσμα της πράξης **next('%')** είναι ο χαρακτήρας '&'

Άλλες πράξεις που περιλαμβάνουν χαρακτήρες

Αντίθετα υπάρχουν αρκετές ανοιχτές πράξεις που περιλαμβάνουν χαρακτήρες είτε ως δεδομένα, είτε ως αποτελέσματα. Μερικές από τις πιο γνωστές είναι:

- **Ord()** : εφαρμόζεται πάνω σε ένα *χαρακτήρα* και μας επιστρέφει έναν *ακέραιο αριθμό* ο οποίος δηλώνει την *θέση* του συγκεκριμένου χαρακτήρα σύμφωνα με το ισχύον σύστημα κωδικοποίησης.
Πχ για την κωδικοποίηση ASCII, το αποτέλεσμα του **Ord('a')** είναι **97**.
- **Char()** : εφαρμόζεται πάνω σε ένα *ακέραιο αριθμό* και μας επιστρέφει έναν *χαρακτήρα* ο οποίος *αντιστοιχεί σε αυτόν τον αριθμό* σύμφωνα με το ισχύον σύστημα κωδικοποίησης.
Πχ για την κωδικοποίηση ASCII, το αποτέλεσμα του **Char(97)** είναι. **'a'**
- *Πράξεις επάνω σε συμβολοσειρές (strings)*. *δημιουργία* συμβολοσειρών από χαρακτήρες, *προσάρτηση*, *αναζήτηση*, *εισαγωγή*, *διαγραφή* χαρακτήρα σε συμβολοσειρά κλπ.

Κωδικοποίηση χαρακτήρων

- Αντιστοίχιση του χαρακτήρα με έναν *μοναδικό θετικό ακέραιο*, ώστε να είναι εφικτή η αποθήκευσή του στη μνήμη του Η/Υ. (πχ. "!" ⇨ 33 "f" ⇨ 102)
- Η πιο διαδεδομένη *μέχρι πρόσφατα* και ιστορικά πρώτη κωδικοποίηση (από το 1963 ως ASA X3.4) ονομάζεται κώδικας *ASCII* (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)
- Ο κώδικας ASCII διαθέτει *256* θέσεις και επομένως χρειάζεται *1 byte* (8 bits) για την αποθήκευση κάθε χαρακτήρα ($2^8=256$).
- Οι πρώτες *32* θέσεις (*0-31*) περιέχουν *ειδικούς χαρακτήρες ελέγχου* οι οποίοι δεν εμφανίζονται στην οθόνη (Enter, Backspace, Tab, Esc).
- Οι επόμενες *96* θέσεις (*32-127*) περιέχουν το *Αγγλικό αλφάβητο* (κεφαλαία και πεζά) , τα ψηφία *0-9* και ένα πλήθος από *ειδικά σύμβολα* όπως αυτά που αναφέραμε νωρίτερα.

Κωδικοποίηση χαρακτήρων

- Για τις τελευταίες 128 θέσεις (128-255) υπάρχει *δυνατότητα επιλογής* των χαρακτήρων που επιθυμούμε να κωδικοποιήσουμε. (πχ στην *Ελλάδα*, θα θέλαμε να κωδικοποιήσουμε τα *Ελληνικά*, στην *Ρωσία* τα *Ρωσικά* κοκ.)
- Αυτό δημιουργεί πολλές *εναλλακτικές μορφές του κώδικα ASCII* που ονομάζονται *κωδικοσελίδες* πχ.
 - *codepage 1250 – Central European*
 - *codepage 1251 – Cyrillic*
 - *codepage 1252 – Western (Latin 1)*
 - *codepage 1253 – Greek*
 - *codepage 1254 – Turkish*
 - *codepage 1255 – Hebrew*
 - *codepage 1256 – Arabic*
 - ...

Κωδικοσελίδες 1253 και 1251 του κώδικα ASCII

Codepage 1253 - Greece Windows

| | -0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -A | -B | -C | -D | -E | -F |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0- | | | | | | | | | | | | | | | | |
| 1- | | | | | | | | | | | | | | | | |
| 2- | | ! | " | # | \$ | % | & | ' | (|) | * | + | , | - | . | / |
| 3- | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4- | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5- | P | Q | R | S | T | U | V | W | X | Y | Z | [| \ |] | ^ | _ |
| 6- | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7- | p | q | r | s | t | u | v | w | x | y | z | { | | } | ~ | |
| 8- | € | ‚ | ƒ | „ | … | † | ‡ | | ‰ | < | | | | | | |
| 9- | ‘ | ’ | “ | ” | • | — | | ™ | > | | | | | | | |
| A- | ‚ | £ | ¤ | ¥ | ¦ | § | ¨ | © | ª | « | ¬ | ® | ¯ | | | |
| B- | ° | ± | ² | ³ | ´ | µ | ¶ | · | ¸ | ¹ | º | » | ¼ | ½ | ¾ | Ω |
| C- | ı | Α | Β | Γ | Δ | Ε | Ζ | Η | Θ | Ι | Κ | Λ | Μ | Ν | Ξ | Ο |
| D- | Π | Ρ | | Σ | Τ | Υ | Φ | Χ | Ψ | Ω | İ | ÿ | ά | έ | ή | ί |
| E- | ϑ | α | β | γ | δ | ε | ζ | η | θ | ι | κ | λ | μ | ν | ξ | ο |
| F- | π | ρ | ς | σ | τ | υ | φ | χ | ψ | ω | ϊ | ϋ | ό | ύ | ώ | |

(the Unicode equivalent code is displayed under each character)

Codepage 1251 - Cyrillic Windows

| | -0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -A | -B | -C | -D | -E | -F |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0- | | | | | | | | | | | | | | | | |
| 1- | | | | | | | | | | | | | | | | |
| 2- | | ! | " | # | \$ | % | & | ' | (|) | * | + | , | - | . | / |
| 3- | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4- | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5- | P | Q | R | S | T | U | V | W | X | Y | Z | [| \ |] | ^ | _ |
| 6- | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7- | p | q | r | s | t | u | v | w | x | y | z | { | | } | ~ | |
| 8- | Ђ | Ѓ | ‚ | ѓ | „ | … | † | ‡ | € | ‰ | Љ | < | Њ | Ќ | Ѕ | Ї |
| 9- | ђ | ‘ | ’ | “ | ” | • | — | | ™ | љ | > | њ | ќ | ћ | џ | |
| A- | Ў | ў | Ј | Ѡ | Ј | Љ | Њ | Ћ | Ќ | © | € | « | ¬ | ® | İ | |
| B- | ° | ± | І | і | г | р | ¶ | · | ё | № | € | » | ј | Ѕ | ѕ | ї |
| C- | А | Б | В | Г | Д | Е | Ж | З | И | Й | К | Л | М | Н | О | П |
| D- | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ъ | Ы | Ь | Э | Ю | Я |
| E- | а | б | в | г | д | е | ж | з | и | й | к | л | м | н | о | п |
| F- | р | с | т | у | ф | х | ц | ч | ш | щ | ъ | ы | ь | э | ю | я |

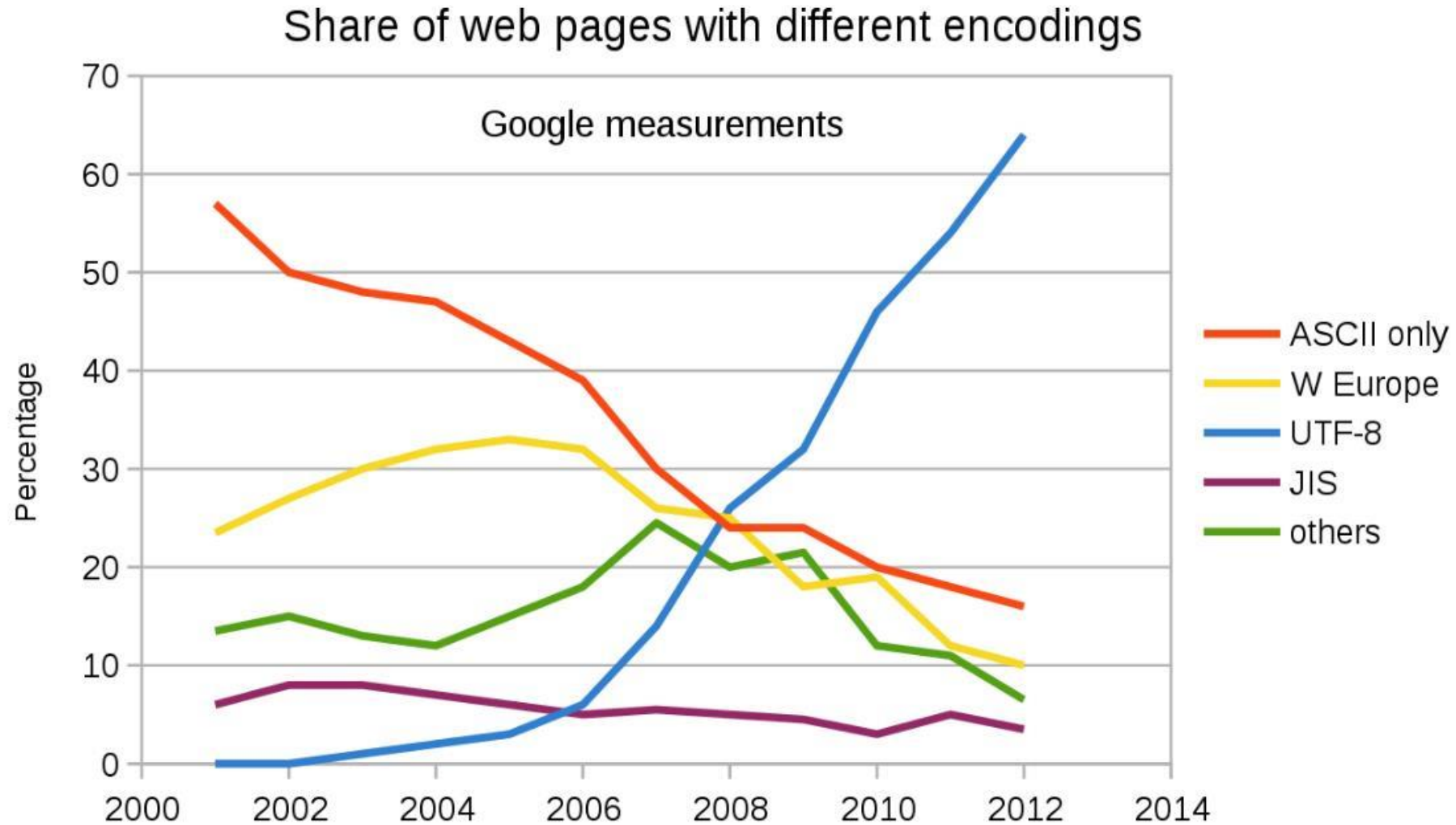
(the Unicode equivalent code is displayed under each character)

- Βασικός **περιορισμός** του κώδικα ASCII είναι ότι **δεν υποστηρίζει πολύγλωσσο κείμενο** (πχ . δεν μπορούμε να έχουμε ταυτόχρονα Ελληνικά και Ρωσικά λόγω διαφορετικών κωδικοσελίδων)

Αρχεία κειμένου και κωδικοποίηση χαρακτήρων

- Για πραγματικά *πολύγλωσσο κείμενο*, χρειαζόμαστε μια κωδικοποίηση με *αρκετές θέσεις* ώστε να περιλαμβάνει τα *αλφάβητα όλων των γλωσσών του κόσμου* (ή περίπου...)
- Η πιο γνωστή κωδικοποίηση που υποστηρίζει πολύγλωσσο κείμενο ονομάζεται *Unicode* και *τείνει να γίνει το νέο παγκόσμιο στάνταρ*, αντικαθιστώντας την κωδικοποίηση ASCII
- Με την κωδικοποίηση *Unicode* σε κάθε χαρακτήρα αντιστοιχεί ένας μοναδικός ακέραιος θετικός αριθμός που ονομάζεται *code point*. Σε αντίθεση όμως με τον ASCII δεν υπάρχει άμεση αντιστοίχιση αυτού του αριθμού σε ένα ή περισσότερα bytes (*octets – οκτάδες από bits*)
- Αυτό γίνεται έτσι ώστε να δημιουργηθεί ένα *παγκόσμιο στάνταρ κωδικοποίησης (ενας μοναδικός αριθμός [code point] για κάθε σύμβολο ή χαρακτήρα οποιασδήποτε γλώσσας)* το οποίο να μπορεί να κωδικοποιηθεί σε *octets* με διάφορους τρόπους
- Ανάλογα με τον *τρόπο αντιστοίχισης των code points σε octets*, υπάρχουν και διαφορετικές παραλλαγές του κώδικα Unicode.
- Η πιο *γνωστή και διαδεδομένη* παραλλαγή του κώδικα Unicode ονομάζεται *UTF-8 (8-bit Unicode Transformation Format)*. Υπάρχουν και άλλες δυο λιγότερο διαδεδομένες παραλλαγές που ονομάζονται *UTF-16* και *UTF-32*

Η κωδικοποίηση UTF-8 τείνει να γίνει το νέο παγκόσμιο στάνταρ



Αρχεία κειμένου και κωδικοποίηση χαρακτήρων

- Στην κωδικοποίηση *UTF-8* όλοι οι χαρακτήρες *δεν κωδικοποιούνται με τον ίδιο αριθμό bytes* (*variable length encoding*)
- Οι χαρακτήρες με τη *μεγαλύτερη συχνότητα* εμφάνισης κωδικοποιούνται σε *ένα byte* (octet). *Οι χαρακτήρες αυτοί αντιστοιχούν εξ' ορισμού με τους πρώτους 128 χαρακτήρες του κώδικα ASCII. Έτσι, για κείμενα που περιέχουν μόνο Αγγλικά, οι κώδικες utf-8 και ASCII ταυτίζονται.*
- Οι επόμενοι *1920* χαρακτήρες κωδικοποιούνται σε *δυο bytes* και περιλαμβάνουν τα υπόλοιπα ειδικά σύμβολα όλων των Λατινογενών αλφαβήτων (*ë, î, ģ, ï* κλπ.), τα *Ελληνικά, Κυριλλικά, Αρμενικά, Εβραϊκά, Συριακά αλφάβητα* κ.α.
- Τα σύμβολα και οι χαρακτήρες των υπολοίπων γλωσσών του κόσμου (*Κινέζικα, Ιαπωνικά, Κορεάτικα* κλπ.) κωδικοποιούνται σε *3 ή 4 bytes*.
- Πρακτικά για μας αυτό σημαίνει ότι αν ένα *αρχείο* περιέχει κείμενο σε *κωδικοποίηση UTF-8*
 - Όλοι οι *Αγγλικοί χαρακτήρες* και τα ειδικά σύμβολα καταλαμβάνουν *1 byte* στο αρχείο
 - Όλοι οι *Ελληνικοί χαρακτήρες* καταλαμβάνουν *2 bytes* στο αρχείο

Παραδείγματα αρχείων με διαφορετική κωδικοποίηση

```
>>> f = open("C:/Users/MK/Desktop/Petros.txt", "w", encoding = "cp1253")
>>> f.write("Hello Πέτρο!")
12
>>> f.close()
```

| Byte | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|-----------|---|---|---|---|---|---|---|---|---|----|----|----|
| Character | H | e | l | l | o | | Π | έ | τ | ρ | ο | ! |

```
>>> f = open("C:/Users/MK/Desktop/Petros.txt", "w", encoding = "utf-8")
>>> f.write("Hello Πέτρο!")
12
>>> f.close()
```

| Byte | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|-----------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|
| Character | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 8 | | 9 | | 10 | | 11 | | 12 |
| | H | e | l | l | o | | Π | | έ | | τ | | ρ | | ο | | ! |

```
>>> f = open("C:/Users/MK/Desktop/Petros.txt", encoding = "utf-8")
>>> f.readline()
'Hello Πέτρο!'
```

```
>>> f.seek(5)
5
>>> f.read(1)
' '
>>> f.tell()
6
>>> f.read(1)
'Π'
>>> f.tell()
8
>>> f.read(1)
'έ'
>>> f.tell()
10
```

Παραδείγματα αρχείων με διαφορετική κωδικοποίηση

```
>>> f = open("C:/Users/MK/Desktop/Petros_2.txt", "w", encoding = "utf-8")
>>> f.write("Hello Πέτρο Πιτερ!")
18
>>> f.close()
```

| Byte | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
|-----------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Character | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| | H | e | l | l | o | | Π | έ | τ | ρ | ο | | Π | ι | τ | e | p | ! | | | | | | | | | | |

```
>>> f = open("C:/Users/MK/Desktop/Petros_2.txt", "r", encoding = "utf-8")
>>> f.read()
'Hello Πέτρο Πιτερ!'
>>> f.seek(19)
19
>>> f.read(1)
'ι'
>>> f.seek(22)
22
>>> f.read(1)
Traceback (most recent call last):
UnicodeDecodeError: 'utf-8' codec can't decode byte 0x82
in position 0: invalid start byte
```

Λογικοί (Logical ή Boolean)

- Ο τύπος αυτός περιέχει τις τιμές που μπορεί να πάρει μια *λογική πρόταση*.
- Αυτές είναι δυο όλες κι όλες. Η τιμή *true* (αλήθεια / σωστό) και η τιμή *false* (ψέματα / λάθος) .
- Οι πιο συνηθισμένες πράξεις που εφαρμόζονται πάνω σε λογικές τιμές και παράγουν επίσης λογικά αποτελέσματα είναι οι *AND* (λογικό και), *OR* (λογικό είτε), *NOT* (λογική άρνηση) και *XOR* (λογικό αποκλειστικό είτε).

| Τιμή πρώτης μεταβλητής | Τιμή δεύτερης μεταβλητής | Λογική πράξη | | | |
|------------------------|--------------------------|--------------|-------|--------|-------|
| | | AND | OR | NOT(*) | XOR |
| True | True | True | True | False | False |
| True | False | False | True | False | True |
| False | True | False | True | True | True |
| False | False | False | False | True | False |

(*) Εφαρμογή στην πρώτη μεταβλητή μόνο

Τελεστές σύγκρισης

Τελεστές που εφαρμόζονται *σε άλλες κατηγορίες τύπων δεδομένων* (ακέραιοι, χαρακτήρες...) και *παράγουν λογικούς αριθμούς*.

Οι πιο συνηθισμένοι τελεστές σύγκρισης είναι οι παρακάτω

- `<` μικρότερο
- `<=` μικρότερο ή ίσο
- `>` μεγαλύτερο
- `>=` μεγαλύτερο ή ίσο
- `<>` διάφορο (όχι ίσο)
- `==` ίσον

Παραδείγματα

`18 <= 5` \Rightarrow `False`

`'D' < 'F'` \Rightarrow `True` επειδή το `'D'` προηγείται του `'F'` στην κωδικοποίηση ASCII

Σειρές Χαρακτήρων (συμβολοσειρές / strings)

- Σειρές (strings) *διαδοχικών χαρακτήρων* / συμβόλων που αποτελούν *αυτόνομες ενότητες*
- Σύμφωνα με τον ορισμό αυτό, κάθε *λέξη*, *πρόταση*, γενικά κάθε *κείμενο* αποτελεί μια *συμβολοσειρά* (string) .
- Οι σειρές χαρακτήρων *δεν μπορούν να χαρακτηριστούν* ακριβώς *στοιχειώδης τύπος* δεδομένων αφού αποτελούνται από δεδομένα *απλούστερου* τύπου
- Συγκαταλέγονται κάπου *ενδιάμεσα* στους *απλούς* και τους *δομημένους* τύπους δεδομένων, επειδή τα δομικά τους στοιχεία αντλούνται αποκλειστικά από ένα και μόνο τύπο (των χαρακτήρων)
- Οι σειρές χαρακτήρων στις γλώσσες προγραμματισμού συνήθως περικλείονται με *διπλά* ή *μονά εισαγωγικά* (αγγλ. *quote*) :

"αυτή η σειρά χαρακτήρων έχει διπλά εισαγωγικά στα άκρα της"

'βάζοντας στα άκρα μονά εισαγωγικά, περικλείουμε το διπλό " χωρίς πρόβλημα'

Εσωτερική αναπαράσταση σειράς χαρακτήρων

Διαφορετικές γλώσσες προγραμματισμού υλοποιούν διαφορετικούς τρόπους αποθήκευσης σειρών χαρακτήρων στην μνήμη του υπολογιστή.

1. *Length prefixed* {μήκος, σειρά χαρακτήρων}

- Της σειράς χαρακτήρων προηγείται ένας *ακέραιος* ο οποίος δηλώνει το *μήκος της*
- Χρησιμοποιείται μεταξύ άλλων και στις γλώσσες προγραμματισμού Pascal και FORTRAN

| | | | | | |
|------------------|------------------|------------------|------------------|------------------|------------------|
| length | F | R | A | N | K |
| 05 ₁₆ | 46 ₁₆ | 52 ₁₆ | 41 ₁₆ | 4E ₁₆ | 4B ₁₆ |

2. *Null-terminated* {σειρά χαρακτήρων, ειδικός χαρακτήρας τερματισμού NULL}

- Η σειρά χαρακτήρων τερματίζει με τον ειδικό χαρακτήρα NULL (κωδικός 00)
- Χρησιμοποιείται μεταξύ άλλων στη γλώσσα προγραμματισμού C

| | | | | | |
|------------------|------------------|------------------|------------------|------------------|------------------|
| F | R | A | N | K | NUL |
| 46 ₁₆ | 52 ₁₆ | 41 ₁₆ | 4E ₁₆ | 4B ₁₆ | 00 ₁₆ |

Εσωτερική αναπαράσταση σειράς χαρακτήρων

3. *Byte-terminated* {σειρά χαρακτήρων, ειδικός χαρακτήρας τερματισμού}

- Παλαιότερη μέθοδος αναπαράστασης – *τείνει να καταργηθεί*
- Η σειρά χαρακτήρων τερματίζει με κάποιον *ειδικό χαρακτήρα τερματισμού*, ο οποίος *δεν επιτρέπεται να συμπεριλαμβάνεται* σε αυτήν.
- Ο χαρακτήρας που χρησιμοποιείται συνήθως είναι το *\$*

4. *Bit-terminated* {σειρά χαρακτήρων *πλην τελευταίου*, *τελευταίος χαρακτήρας* σειράς}

- Παλαιότερη μέθοδος αναπαράστασης – *δεν χρησιμοποιείται πλέον*
- Είχε εφαρμογή τότε που ο κώδικας ASCII περιελάμβανε μόνο *128* χαρακτήρες, επομένως χρειαζόταν *μόνο 7* από τα *8* bits ενός byte για την υλοποίησή του
- Το 8^ο bit είχε την τιμή *0* για όλους τους χαρακτήρες της σειράς, *εκτός του τελευταίου* που έπαιρνε την τιμή *1*, σηματοδοτώντας έτσι το τέλος της.

Πράξεις πάνω σε σειρές χαρακτήρων

Υλοποιούνται συνήθως υπό μορφή συναρτήσεων. Τυπικό παράδειγμα: C++

| Functions | Description |
|--------------------------|--|
| strlen(s1) | Returns the length of a string s1 |
| strcpy(s1,s2) | Copy a string s2 into string s1 |
| strncpy(s1,s2) | Copies character of a string s2 to another string s1 up to the specified length n. |
| strcmp(s1,s2) | Compare two string s1 and s2 (Function discriminates between upper case and lower case) Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2 |
| stricmp(s1,s2) | Compare two string s1 and s2 (Function doesn't discriminates between upper case and lower case) Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2 |
| strncmp(s1,s2,n) | Compares character of two string s1 and s2 upto a specified length. |
| strnicmp(s1,s2,n) | Compares characters of two string s1 and s2 upto a specified length. Ignore case. |
| strlwr(s1) | Converts the uppercase character of string s1 to lowercase. |
| strupr(s1) | Converts the lowercase character of string s1 to uppercase. |
| strdup(s1) | This function duplicates the string s1 at the allocated memory which is pointed by a pointer variable. |
| strchr(s1,c) | Returns a pointer to the first occurrence of character c in string s1. |
| strrchr(s1,c) | Returns a pointer to the last occurrence of character c in string s1. |
| strcat(s1,s2) | Concatenates string s2 onto the end of string s1. |
| strncat(s1,s2,n) | Concates strings s1 and s2 upto a specified length n. |
| strrev(s1) | Reverse all the characters of string s1. |
| strstr(s1, s2); | Returns a pointer to the first occurrence of string s2 in string s1. |

Σύνθετοι ή Δομημένοι Τύποι δεδομένων (Structured Data Types)

Παράγονται από τους *απλούς τύπους δεδομένων* και χρησιμοποιούνται για την *αποθήκευση* και *επεξεργασία* δεδομένων που έχουν *πιο σύνθετη δομή*.

Χαρακτηριστικό τους η *μεγάλη ευελιξία στη σύνθεσή τους* με δομικά στοιχεία είτε *απλούς*, είτε άλλους προκατασκευασμένους *δομημένους τύπους δεδομένων*

Οι πιο διαδεδομένοι δομημένοι τύποι δεδομένων είναι:

- Διανύσματα / Πίνακες (Arrays / Tables)
- Εγγραφές (Records)
- Λίστες (Lists)
- Αρχεία (Files)
- Σύνολα (Sets)
- Λεξικά (Dictionaries)

Διανύσματα / Πίνακες (Arrays / Tables)

- Διάνυσμα: *πεπερασμένο, διατεταγμένο* σύνολο από στοιχεία του *ίδιου τύπου*, κάτω από ένα *κοινό όνομα*
- Ο *εντοπισμός* ενός συγκεκριμένου στοιχείου κάποιου διανύσματος γίνεται με την βοήθεια του *δείκτη (index)* που είναι ένας *ακέραιος θετικός αριθμός* ο οποίος προσδιορίζει την ακριβή θέση του στοιχείου μέσα στο διάνυσμα
- Ο δείκτης γράφεται συνήθως *σε παρένθεση δίπλα στο όνομα* του διανύσματος
- Τα *στοιχεία ενός διανύσματος* μπορεί να είναι ακέραιοι, πραγματικοί, χαρακτήρες ή γενικότερα *οποιοδήποτε* άλλου *απλού ή σύνθετου τύπου*
- Επειδή όλα τα στοιχεία ενός διανύσματος είναι *ίδιου τύπου*, κάθε στοιχείο καταλαμβάνει τον *ίδιο αριθμό από bytes* όταν αποθηκεύεται στη μνήμη του Η/Υ

Μονοδιάστατα διανύσματα (one dimensional arrays)

- Η πιο απλή μορφή ενός διανύσματος είναι το μονοδιάστατο διάνυσμα.
- Ένα μονοδιάστατο διάνυσμα 10 θέσεων με όνομα A θα μπορούσε να παρασταθεί ως εξής:

| | | | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|--------|
| A (1) | A (2) | A (3) | A (4) | A (5) | A (6) | A (7) | A (8) | A (9) | A (10) |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|--------|

Συνάρτηση απεικόνισης διανύσματος (array mapping function)

| | |
|-----|-------|
| 100 | A(1) |
| 101 | |
| 102 | A(2) |
| 103 | |
| 104 | A(3) |
| 105 | |
| 106 | A(4) |
| 107 | |
| 108 | A(5) |
| 109 | |
| 110 | A(6) |
| 111 | |
| 112 | A(7) |
| 113 | |
| 114 | A(8) |
| 115 | |
| 116 | A(9) |
| 117 | |
| 118 | A(10) |
| 119 | |

- Όταν ένα διάνυσμα αποθηκεύεται στη μνήμη του ηλεκτρονικού υπολογιστή, η αποθήκευση γίνεται σε *διαδοχικές* θέσεις της μνήμης.
- Ανάλογα με τον τύπο του διανύσματος, κάθε στοιχείο του απαιτεί *ένα συγκεκριμένο αριθμό θέσεων μνήμης (bytes)* για την αποθήκευσή του.
- Π.χ. για ένα διάνυσμα ακεραίων με διάσταση 10, κάθε στοιχείο του απαιτεί για την αποθήκευσή του 2 θέσεις μνήμης (2 bytes) οπότε καταλαμβάνει συνολικά 20 bytes
- Η διεύθυνση μνήμης από την οποία ξεκινάει η αποθήκευση ενός διανύσματος ονομάζεται *Διεύθυνση Βάσης [ΔΒ]* (Base Address) (στο παράδειγμά μας ΔΒ=100)
- Η τιμή της ΔΒ εξαρτάται από το Λειτουργικό Σύστημα και είναι εν γένει διαφορετική κάθε φορά

Συνάρτηση απεικόνισης διανύσματος (array mapping function)

| | |
|-----|-------|
| 100 | A(1) |
| 101 | |
| 102 | A(2) |
| 103 | |
| 104 | A(3) |
| 105 | |
| 106 | A(4) |
| 107 | |
| 108 | A(5) |
| 109 | |
| 110 | A(6) |
| 111 | |
| 112 | A(7) |
| 113 | |
| 114 | A(8) |
| 115 | |
| 116 | A(9) |
| 117 | |
| 118 | A(10) |
| 119 | |

Πως μπορούμε να βρούμε από ποια θέση μνήμης αρχίζει η αποθήκευση του στοιχείου $A(j)$ αν γνωρίζουμε

1. τον δείκτη j του στοιχείου
2. την διεύθυνση βάσης $\Delta B = 100$ και
3. το μέγεθος $M = 2$ (πλήθος θέσεων μνήμης που καταλαμβάνει κάθε στοιχείου)

Σκεφτόμαστε ως εξής

- Το στοιχείο $A(1)$ αποθηκεύεται ξεκινώντας από την διεύθυνση ΔB
- Το στοιχείο $A(2)$ αποθηκεύεται ξεκινώντας από την διεύθυνση $\Delta B + M$
- Το στοιχείο $A(3)$ αποθηκεύεται ξεκινώντας από την διεύθυνση $\Delta B + 2 * M$
- Το στοιχείο $A(4)$ αποθηκεύεται ξεκινώντας από την διεύθυνση $\Delta B + 3 * M$

και γενικότερα

- Το στοιχείο $A(j)$ αποθηκεύεται ξεκινώντας από την διεύθυνση $\Delta B + (j-1) * M$

Συνάρτηση απεικόνισης διανύσματος (array mapping function)

| | |
|-----|-------|
| 100 | A(1) |
| 101 | |
| 102 | A(2) |
| 103 | |
| 104 | A(3) |
| 105 | |
| 106 | A(4) |
| 107 | |
| 108 | A(5) |
| 109 | |
| 110 | A(6) |
| 111 | |
| 112 | A(7) |
| 113 | |
| 114 | A(8) |
| 115 | |
| 116 | A(9) |
| 117 | |
| 118 | A(10) |
| 119 | |

Συνάρτηση απεικόνισης μονοδιάστατου διανύσματος για τον υπολογισμό της θέσης του στοιχείου $A(j)$ ενός μονοδιάστατου διανύσματος, αποθηκευμένου στη μνήμη όταν γνωρίζουμε την διεύθυνση βάσης ΔB και το μέγεθος M του κάθε στοιχείου του

Συνάρτηση Απεικόνισης Μονοδιάστατου Διανύσματος

$$f(j, M, \Delta B) = \Delta B + (j - 1) * M$$

Εφαρμογή. Να βρεθεί η θέση του 5^{ου} στοιχείου ενός μονοδιάστατου διανύσματος ακεραίων ($M=2$) που είναι αποθηκευμένο στη μνήμη ενός Η/Υ αρχίζοντας από την θέση 300.

$$f(5, 2, 300) = 300 + (5 - 1) * 2 = 308.$$

Δισδιάστατοι πίνακες / διανύσματα (two dimensional arrays)

- Ένα *δισδιάστατο* διάνυσμα, είναι ένα διάνυσμα, *κάθε στοιχείο του οποίου* είναι ένα άλλο διάνυσμα
- Συνήθως εδώ χρησιμοποιούμε περισσότερο τον όρο *πίνακας*, αν και ο όρος διάνυσμα είναι απόλυτα ισοδύναμος

| | | | |
|--------|--------|--------|--------|
| B(1,1) | B(1,2) | B(1,3) | B(1,4) |
| B(2,1) | B(2,2) | B(2,3) | B(2,4) |
| B(3,1) | B(3,2) | B(3,3) | B(3,4) |
| B(4,1) | B(4,2) | B(4,3) | B(4,4) |
| B(5,1) | B(5,2) | B(5,3) | B(5,4) |

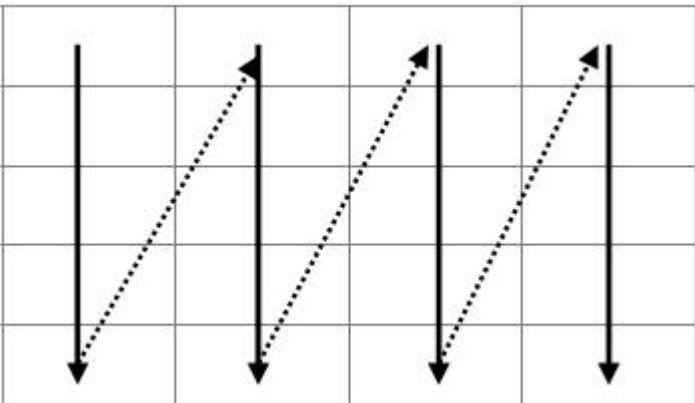
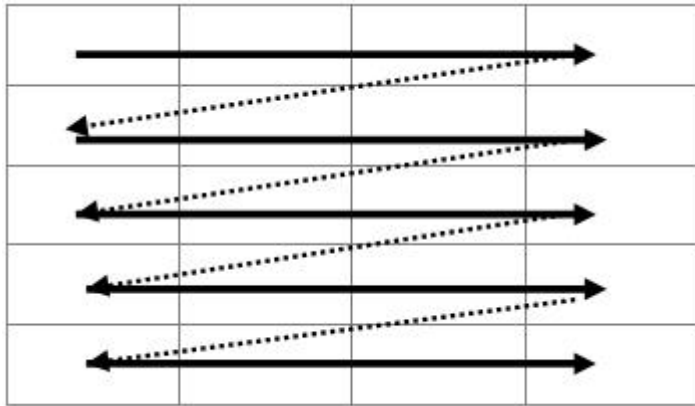
Παράδειγμα: δισδιάστατος πίνακας *5x4*
(5 γραμμές, 4 στήλες)

Για να υπολογίσουμε τη *συνάρτηση απεικόνισής του* χρειάζεται να γνωρίζουμε τον *τρόπο αποθήκευσής του στην σειριακή* (και επομένως εξ ορισμού μονοδιάστατη) *μνήμη* του υπολογιστή.

- Πως απεικονίζουμε μια πολυδιάστατη δομή σε μια μονοδιάστατη (σειριακή);

Συνάρτηση απεικόνισης δισδιάστατου πίνακα

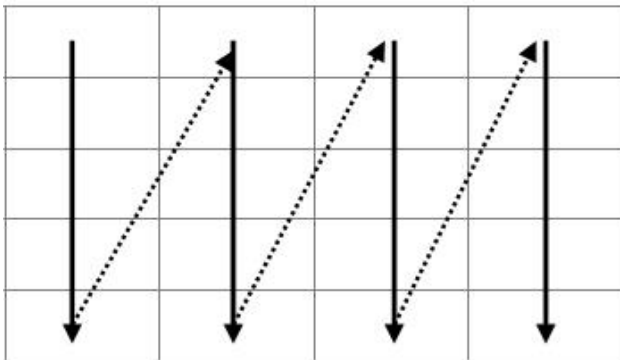
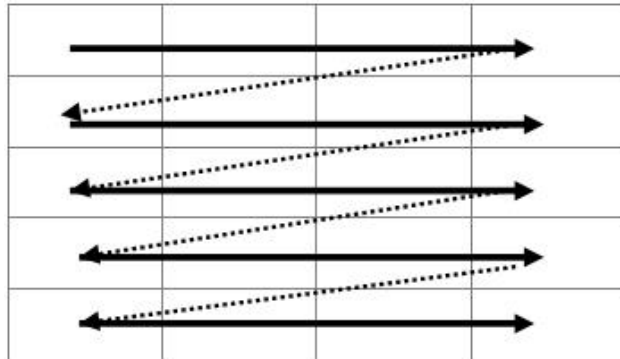
Εξαρτάται από τον τρόπο αποθήκευσης το πίνακα που υποστηρίζεται από την εκάστοτε γλώσσα προγραμματισμού. Υπάρχουν δυο τεχνικές αποθήκευσης.



- **Αποθήκευση κατά γραμμές:** Αποθηκεύουμε την *πρώτη γραμμή* του πίνακα στην μνήμη σαν να επρόκειτο για ένα *μονοδιάστατο διάνυσμα*. Στη συνέχεια αποθηκεύουμε την δεύτερη γραμμή, μετά την τρίτη κ.ο.κ. μέχρι να αποθηκεύσουμε και την τελευταία γραμμή.
- **Αποθήκευση κατά στήλες:** Αποθηκεύουμε την *πρώτη στήλη* του πίνακα στην μνήμη σαν να επρόκειτο για ένα *μονοδιάστατο διάνυσμα*. Στη συνέχεια αποθηκεύουμε την δεύτερη στήλη, μετά την τρίτη κ.ο.κ. μέχρι να αποθηκεύσουμε και την τελευταία στήλη.

Συνάρτηση απεικόνισης δισδιάστατου πίνακα

Η συνάρτηση απεικόνισης εδώ έχει δυο μεταβλητές i και j δεδομένου ότι μας ενδιαφέρει η θέση του στοιχείου $B(i, j)$. Οι υπόλοιπες - γνωστές - παράμετροι είναι



- ΔB : η διεύθυνση βάσης(πχ 500)
- M : το μέγεθος των κελιών (πχ. 2)
- i_max : η μέγιστη διάσταση του i (πχ. 5)
- j_max : η μέγιστη διάσταση του j (πχ. 4)

| | (α) |
|-----|----------|
| 500 | B (1, 1) |
| 502 | B (1, 2) |
| 504 | B (1, 3) |
| 506 | B (1, 4) |
| 508 | B (2, 1) |
| 510 | B (2, 2) |
| 512 | B (2, 3) |
| 514 | B (2, 4) |
| 516 | B (3, 1) |
| 518 | B (3, 2) |
| 520 | B (3, 3) |
| 522 | B (3, 4) |
| 524 | B (4, 1) |
| 526 | B (4, 2) |
| 528 | B (4, 3) |
| 530 | B (4, 4) |
| 532 | B (5, 1) |
| 534 | B (5, 2) |
| 536 | B (5, 3) |
| 538 | B (5, 4) |

Αποθήκευση κατά γραμμές

Για να εντοπίσουμε ένα στοιχείο που βρίσκεται στην γραμμή **i**, θα πρέπει αρχικά να προσπεράσουμε τις προηγούμενες **i-1** γραμμές, η καθεμιά από τις οποίες έχει **j_max** στοιχεία, θα πρέπει δηλαδή να μετακινηθούμε **j_max * (i-1)** στοιχεία.

Επειδή κάθε στοιχείο του πίνακα καταλαμβάνει **M** διαδοχικές θέσεις μνήμης, ο συνολικός αριθμός θέσεων μνήμης που θα πρέπει να μετακινηθούμε για να βρεθούμε στην αρχή της γραμμής που μας ενδιαφέρει είναι **j_max * (i-1) * M**

Βρισκόμενοι τώρα στην αρχή της γραμμής **i**, θα πρέπει να προσπεράσουμε και τα **j-1** προηγούμενα στοιχεία πάνω στη γραμμή, έτσι ώστε να βρεθούμε τελικά στο στοιχείο **j**. Αυτό σημαίνει ότι θα πρέπει να μετακινηθούμε **(j-1) * M** θέσεις ακόμα πιο κάτω

Προσθέτοντας αυτή την μετακίνηση στην αρχική μας και λαμβάνοντας υπόψη την διεύθυνση βάσης **ΔB** απ' όπου οφείλουμε να ξεκινήσουμε, προκύπτει ο τύπος για τη συνάρτηση απεικόνισης δισδιάστατου πίνακα αποθηκευμένου κατά γραμμές.

$$f(i, j, j_max, M, \Delta B) = \Delta B + (j_max) * (i-1) * M + (j-1) * M$$

| | (α) |
|-----|----------|
| 500 | B (1, 1) |
| 502 | B (1, 2) |
| 504 | B (1, 3) |
| 506 | B (1, 4) |
| 508 | B (2, 1) |
| 510 | B (2, 2) |
| 512 | B (2, 3) |
| 514 | B (2, 4) |
| 516 | B (3, 1) |
| 518 | B (3, 2) |
| 520 | B (3, 3) |
| 522 | B (3, 4) |
| 524 | B (4, 1) |
| 526 | B (4, 2) |
| 528 | B (4, 3) |
| 530 | B (4, 4) |
| 532 | B (5, 1) |
| 534 | B (5, 2) |
| 536 | B (5, 3) |
| 538 | B (5, 4) |

Αποθήκευση κατά γραμμές

$$f(i, j, j_max, M, \Delta B) = \Delta B + (j_max) * (i-1) * M + (j-1) * M$$

Εφαρμογή.

Ένας δισδιάστατος πίνακας ακεραίων B(5x4) είναι αποθηκευμένος κατά γραμμές στη μνήμη ενός ηλεκτρονικού υπολογιστή αρχίζοντας από την θέση 500. Βρείτε την θέση μνήμης απ' όπου ξεκινά η αποθήκευση του στοιχείου B(3,4).

Δεδομένα: $\Delta B=500$ $M=2$ $i=3$ $j=4$ $i_max=5$ $j_max=4$

Εφαρμόζοντας τα δεδομένα μας στον παραπάνω τύπο παίρνουμε

$$\begin{aligned}
 f(3, 4, 4, 2, 500) &= 500 + 4 * (3-1) * 2 + (4-1) * 2 \\
 &= 500 + 4 * 2 + 6 \\
 &= 522
 \end{aligned}$$

Αποθήκευση κατά στήλες

Ο τύπος προκύπτει κατ' αναλογία με τον προηγούμενο

$$f(i, j, i_max, M, \Delta B) = \Delta B + (i_max) * (j-1) * M + (i-1) * M$$

Εφαρμογή.

Ένας δισδιάστατος πίνακας ακεραίων $B(5 \times 4)$ είναι αποθηκευμένος κατά στήλες στη μνήμη ενός ηλεκτρονικού υπολογιστή αρχίζοντας από την θέση 500. Βρείτε την θέση μνήμης απ' όπου ξεκινά η αποθήκευση του στοιχείου $B(4,2)$.

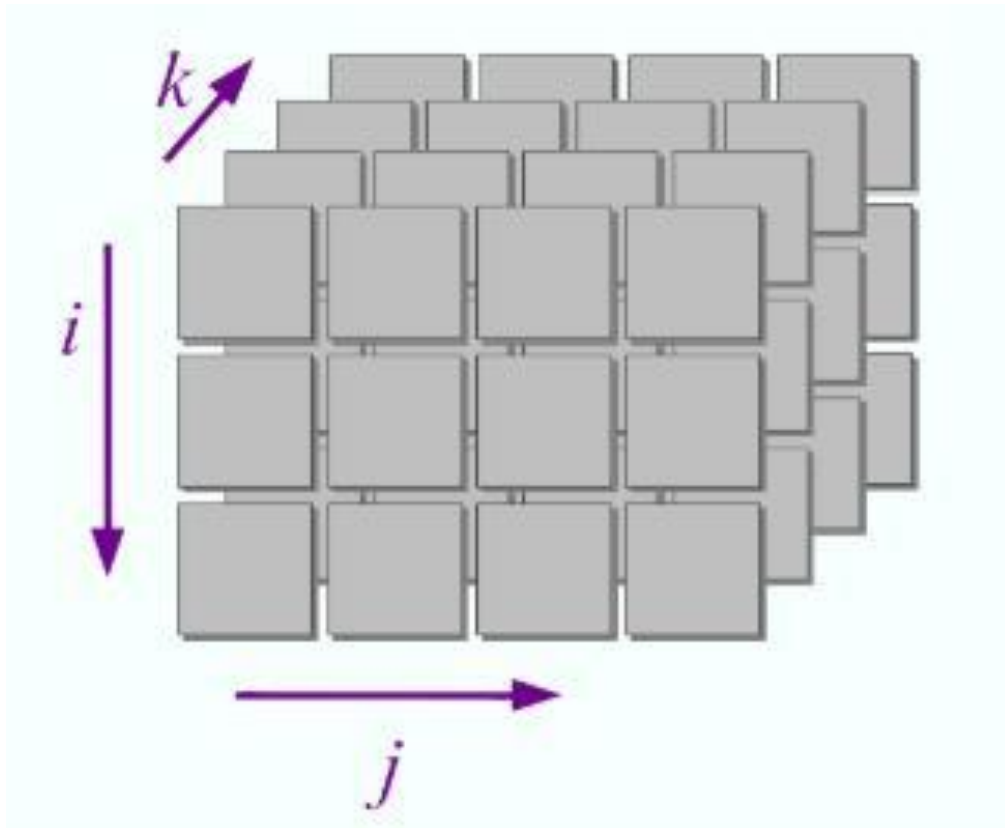
Δεδομένα: $\Delta B=500$ $M=2$ $i=4$ $j=2$ $i_max=5$ $j_max=4$

Εφαρμόζοντας τα δεδομένα μας στον παραπάνω τύπο παίρνουμε

$$\begin{aligned} f(4, 2, 5, 2, 500) &= 500 + 5 * (2-1) * 2 + (4-1) * 2 \\ &= 500 + 5 * 2 + 6 \\ &= 516 \end{aligned}$$

| | (β) |
|-----|--------|
| 500 | B(1,1) |
| 502 | B(2,1) |
| 504 | B(3,1) |
| 506 | B(4,1) |
| 508 | B(5,1) |
| 510 | B(1,2) |
| 512 | B(2,2) |
| 514 | B(3,2) |
| 516 | B(4,2) |
| 518 | B(5,2) |
| 520 | B(1,3) |
| 522 | B(2,3) |
| 524 | B(3,3) |
| 526 | B(4,3) |
| 528 | B(5,3) |
| 530 | B(1,4) |
| 532 | B(2,4) |
| 534 | B(3,4) |
| 536 | B(4,4) |
| 538 | B(5,4) |

Πολυδιάστατοι πίνακες



- Προκύπτουν ως γενίκευση του ορισμού των δισδιάστατων πινάκων
- Ένας *τρισδιάστατος πίνακας* αποθηκεύεται στην μνήμη του ηλεκτρονικού υπολογιστή σαν μια *σειρά δισδιάστατων πινάκων*, κ.ο.κ.
- Οι *συναρτήσεις απεικόνισής τους* ορίζονται με ανάλογο τρόπο

Ειδικές κατηγορίες δισδιάστατων πινάκων

Διαγώνιοι πίνακες.

| | | | | |
|---|---|---|---|---|
| x | 0 | 0 | 0 | 0 |
| 0 | x | 0 | 0 | 0 |
| 0 | 0 | x | 0 | 0 |
| 0 | 0 | 0 | x | 0 |
| 0 | 0 | 0 | 0 | x |

- Πίνακες των οποίων όλα τα στοιχεία που δεν ανήκουν στην *κύρια διαγώνιο* είναι *μηδενικά*
- Εξ' ορισμού *τετραγωνικοί* πίνακες (**$n \times n$**)
- Τυπικός ορισμός: D διαγώνιος $\Leftrightarrow D(i, j) = 0$ για $i \neq j$
- Στη μνήμη αποθηκεύονται *μόνο τα στοιχεία* που βρίσκονται *πάνω στην κύρια διαγώνιο* ως μονοδιάστατος πίνακας.

Ειδικές κατηγορίες δισδιάστατων πινάκων

Τριγωνικοί πίνακες.

L

| | | | | |
|---|---|---|---|---|
| x | 0 | 0 | 0 | 0 |
| x | x | 0 | 0 | 0 |
| x | x | x | 0 | 0 |
| x | x | x | x | 0 |
| x | x | x | x | x |

U

| | | | | |
|---|---|---|---|---|
| x | x | x | x | x |
| 0 | x | x | x | x |
| 0 | 0 | x | x | x |
| 0 | 0 | 0 | x | x |
| 0 | 0 | 0 | 0 | x |

- Κάτω τριγωνικός: ένας τετραγωνικός πίνακας στον οποίο όλα τα στοιχεία πάνω από την κύρια διαγώνιο είναι μηδενικά

$$L(i, j) = \begin{cases} a_{ij} & \text{για } i \leq j \\ 0 & \text{για } i > j \end{cases}$$

- Πάνω τριγωνικός: ένας τετραγωνικός πίνακας στον οποίο όλα τα στοιχεία κάτω από την κύρια διαγώνιο είναι μηδενικά


$$U(i, j) = \begin{cases} a_{ij} & \text{για } i \geq j \\ 0 & \text{για } i < j \end{cases}$$

- Τριγωνικός: ένας πίνακας ο οποίος είναι κάτω τριγωνικός ή πάνω τριγωνικός

Ειδικές κατηγορίες δισδιάστατων πινάκων

Αποθήκευση τριγωνικών πινάκων

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | a | b | c | d |
| 1 | | e | f | g |
| 2 | | | h | i |
| 3 | | | | j |



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- Κατά *γραμμές* ή *στήλες* όπως και οι κανονικοί πίνακες, χωρίς όμως την αποθήκευση των μηδενικών στοιχείων
- Ενας $N \times N$ τριγωνικός πίνακας αποθηκεύεται σε $N \times (N+1) / 2$ διαδοχικές θέσεις μνήμης
- Οι συναρτήσεις απεικόνισης πίνακα προκύπτουν κατ' αναλογία

Παράδειγμα

Η συνάρτηση απεικόνισης ενός *πάνω τριγωνικού* πίνακα αποθηκευμένου *κατά γραμμές* (και χωρίς να λαμβάνεται υπόψη το μήκος λέξης M) είναι:

$$f(i, j, N, \Delta B) = \Delta B + (N * i) + j - ((i * (i + 1)) / 2)$$

Ειδικές κατηγορίες δισδιάστατων πινάκων

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | x | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | x | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | x | 0 | 0 | 0 | x | 0 |
| 0 | 0 | 0 | 0 | x | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Αραιοί Πίνακες (Sparse matrices)

- Δισδιάστατοι πίνακες συνήθως αρκετά μεγάλων διαστάσεων, που περιέχουν μικρό αριθμό μη μηδενικών στοιχείων με τυχαία κατανομή
- Και εδώ επίσης δεν έχει νόημα η αποθήκευση των μηδενικών, οπότε αναπτύσσονται ειδικές τεχνικές αποθήκευσης.
- Η πιο γνωστή από αυτές είναι η λεγόμενη **αποθήκευση τριάδων**.
- για κάθε μη-μηδενικό στοιχείο $s(i, j)$ ενός αραιού πίνακα αποθηκεύεται μια τριάδα τιμών $[i, j, s(i, j)]$, δηλαδή οι δυο δείκτες (*γραμμή, στήλη*) που καθορίζουν τη θέση του στοιχείου μέσα στον πίνακα καθώς και η τιμή του στοιχείου.
- Στις τριάδες που δημιουργούνται, προστίθεται ακόμα μια – συνήθως στην αρχή – στα δυο πρώτα στοιχεία της οποίας αποθηκεύεται η διάσταση του αραιού πίνακα.

Παράδειγμα αποθήκευσης αραιού πίνακα με την μέθοδο των τριάδων

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|----|----|----|----|----|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 45 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 73 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | -5 | 0 | 0 | 0 | 29 | 0 |
| 7 | 0 | 0 | 0 | 0 | 14 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Ο αραιός πίνακας A, παρόλο που έχει διάσταση **8x7** (=56), στην ουσία περιέχει μόνο τις παρακάτω μη-μηδενικές τιμές

- Στη θέση **A(2,3)** την τιμή **45**
- Στη θέση **A(4,4)** την τιμή **73**
- Στη θέση **A(6,2)** την τιμή **-5**
- Στη θέση **A(6,6)** την τιμή **29**
- Στη θέση **A(7,5)** την τιμή **14**

| 1 | 8 | 7 | 0 |
|---|---|---|----|
| 2 | 2 | 3 | 45 |
| 3 | 4 | 4 | 73 |
| 4 | 6 | 2 | -5 |
| 5 | 6 | 6 | 29 |
| 6 | 7 | 5 | 14 |

Επομένως ο πίνακας A μπορεί να 'συμπυκνωθεί' σε ένα πίνακα B μόλις **6x3** (=18), όπου

- η **πρώτη τριάδα** περιέχει την **διάσταση του αρχικού πίνακα** στις στήλες 1 και 2 (και ένα μηδενικό στην τρίτη στήλη)
- σε κάθε μια από τις υπόλοιπες 5 τριάδες περιέχονται αντίστοιχα η **θέση (i,j)** και η **τιμή** των μη-μηδενικών στοιχείων του πίνακα A.

Εγγραφές (Records)

Ομαδοποιημένη συλλογή στοιχείων τα οποία δεν είναι (υποχρεωτικά) του ίδιου τύπου μεταξύ τους. Τα στοιχεία μιας εγγραφής ονομάζονται *πεδία* (fields) ή *μέλη* (members).

- Βασική *διαφορά* ανάμεσα σε πίνακες και εγγραφές
 - ένας *πίνακας* περιέχει πάντα στοιχεία του *ίδιου τύπου*
 - μια *εγγραφή* μπορεί να περιέχει –και κατά κανόνα περιέχει- στοιχεία *διαφορετικού τύπου*.

Αν και διαφοροποιείται ανάλογα με τη γλώσσα προγραμματισμού, ο γενικός τύπος δήλωσης μιας εγγραφής είναι της μορφής

```
TYPE <ΟΝΟΜΑ ΕΓΓΡΑΦΗΣ> = RECORD  
    <ΟΝΟΜΑ ΠΕΔΙΟΥ 1>: <ΤΥΠΟΣ ΠΕΔΙΟΥ 1>;  
    <ΟΝΟΜΑ ΠΕΔΙΟΥ 2>: <ΤΥΠΟΣ ΠΕΔΙΟΥ 2>;  
    ...  
    <ΟΝΟΜΑ ΠΕΔΙΟΥ N>: <ΤΥΠΟΣ ΠΕΔΙΟΥ N>;  
END;
```

Παράδειγμα δήλωσης τύπου εγγραφών

Παραδείγματα δήλωσης ενός τύπου εγγραφής σε *διαφορετικές γλώσσες προγραμματισμού*

| Pascal | FORTRAN | Basic / Visual Basic |
|--|--|---|
| <pre>type PRODUCT = record ID: Integer; Name, Model : String; Cost, Price : Real; end;</pre> | <pre>STRUCTURE /PRODUCT/ INTEGER ID CHARACTER*16 NAME CHARACTER*8 MODEL REAL COST REAL PRICE END STRUCTURE</pre> | <pre>Type PRODUCT ID As Integer; Name, Model As String; Cost, Price As Real; End Type</pre> |

- Η δήλωση των τύπων των εγγραφών γίνεται στην *κορυφή του προγράμματος*, πριν από τη δήλωση των μεταβλητών.
- Κάθε τύπος εγγραφής που δηλώνουμε όπως παραπάνω, αποτελεί έναν *νέο τύπο δεδομένων* για το πρόγραμμά μας, κατά συνέπεια μπορούμε να *δηλώσουμε μια ή περισσότερες μεταβλητές* αυτού του τύπου

Παράδειγμα χρήσης μεταβλητών τύπου εγγραφών

Για παράδειγμα με την επόμενη γραμμή δήλωσης μεταβλητών:

```
CURRENT, PRIOR, STOCK(10) as PRODUCT
```

δηλώνονται 2 μεταβλητές τύπου `PRODUCT` και ένας πίνακας 10 θέσεων τύπου `PRODUCT`

Για να αναφερθούμε σε ένα συγκεκριμένο πεδίο μιας εγγραφής γράφουμε το όνομα της εγγραφής ακολουθούμενο από μια τελεία (.) και το όνομα του πεδίου. Δείτε τα επόμενα παραδείγματα

```
CURRENT.Cost = 120
```

```
PRIOR.Model = 'ZX127'
```

```
TOTAL = STOCK(3).Price + STOCK(8).Price - CURRENT.Price
```

Στο τελευταίο παράδειγμα υποθέστε ότι η **TOTAL** είναι μια απλή μεταβλητή τύπου `Real`.

Λίστες (Lists)

Διατεταγμένες συλλογές από τιμές, όπου η ίδια τιμή μπορεί να εμφανίζεται περισσότερες από μία φορές.

- Κάθε στιγμιότυπο μιας τιμής της λίστας καλείται συνήθως *αντικείμενο* ή *στοιχείο* της λίστας.
- Αν η ίδια τιμή εμφανίζεται *πολλές φορές*, κάθε εμφάνισή της θεωρείται ένα *διακριτό αντικείμενο*.
- Διακρίνουμε δυο βασικούς τύπους από λίστες
 - *Στατικές* Επιτρέπουν μόνο τον έλεγχο και την ενημέρωση των στοιχείων τους
 - *Δυναμικές* (ή συνδεδεμένες) Πέρα από έλεγχο και ενημέρωση, επιτρέπουν ακόμα την εισαγωγή, αντικατάσταση ή εξαγωγή των στοιχείων τους.
- Κάποιες γλώσσες προγραμματισμού παρέχουν υποστήριξη για λίστες ως τύπο δεδομένων, και έχουν ειδική σύνταξη και σημασιολογία αλλά και λειτουργίες πάνω σε αυτές
- Μια λίστα συχνά κατασκευάζεται γράφοντας τα αντικείμενά της στη σειρά, χωρισμένα με κόμμα ή κενό, ανάμεσα σε ένα ζευγάρι οριοθετών όπως παρενθέσεις ή αγκύλες

Χαρακτηριστικές ιδιότητες μιας λίστας

- Το **μέγεθος** της λίστας
 - Υποδεικνύει πόσα στοιχεία περιέχει η λίστα
- Ο **τύπος** των στοιχείων της λίστας
 - Όλα τα στοιχεία μιας λίστας έχουν συνήθως τον ίδιο τύπο, ο οποίος αν και μπορεί να είναι οποιοσδήποτε συνήθως είναι τύπος εγγραφής (record)
- Ο **δείκτης** των στοιχείων της λίστας
 - Κάθε στοιχείο της λίστας έχει κάποιο δείκτη, με το πρώτο στοιχείο να έχει συνήθως δείκτη το **0** ή το **1** (ή κάποιον προκαθορισμένο ακέραιο)
 - **Γειτονικά στοιχεία** έχουν δείκτες που διαφέρουν κατά 1 μονάδα
 - Το **τελευταίο στοιχείο** έχει δείκτη: <αρχικός δείκτης> + <μέγεθος λίστας> – 1.

- Είναι εφικτό να *ανακτήσεις* ένα στοιχείο με συγκεκριμένο δείκτη
- Είναι εφικτό να *διατρέξεις* τη λίστα σε αύξουσα σειρά των δεικτών της
- Είναι εφικτό να *αλλάξεις την τιμή* ενός στοιχείου σε μια συγκεκριμένη θέση, χωρίς να επηρεάσεις τα υπόλοιπα στοιχεία της λίστας
- Είναι εφικτό να *εισάγεις* ένα στοιχείο σε μια συγκεκριμένη θέση. Οι δείκτες των μεγαλύτερων στοιχείων αυξάνονται κατά 1
- Είναι εφικτό να *διαγράψεις* ένα στοιχείο σε μια συγκεκριμένη θέση. Οι δείκτες των μεγαλύτερων στοιχείων μειώνονται κατά 1

Υλοποίηση συνδεδεμένης λίστας

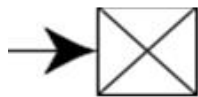
Οι λίστες τυπικά υλοποιούνται είτε ως *συνδεδεμένες λίστες* (απλά ή αμφίδρομα) είτε ως *πίνακες μεταβλητού μεγέθους* (δυναμικοί πίνακες).

Ο πιο συνηθής τρόπος υλοποίησης λιστών (προέρχεται από τη γλώσσα προγραμματισμού *Lisp*) είναι: κάθε στοιχείο της λίστας *πέρα από τα δεδομένα του*, περιέχει και *δείκτη* ή *δείκτες* προς τα γειτονικά του στοιχεία. Αυτό έχει ως αποτέλεσμα τη δημιουργία μίας συνδεδεμένης λίστας



Συνήθως ένας δείκτης αντιστοιχεί στην *διεύθυνση της κεντρικής μνήμης* από την οποία ξεκινάει η αποθήκευση των δεδομένων του στοιχείου της λίστας στο οποίο αναφέρεται (αρχική διεύθυνση μνήμης)

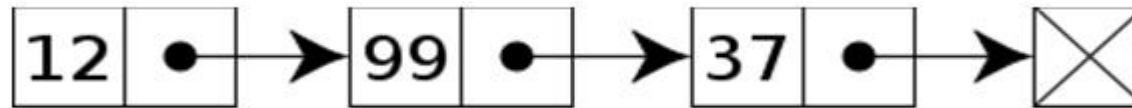
Το τελευταίο στοιχείο μιας λίστας συνήθως συνδέεται σε ένα *ειδικό κόμβο* που ονομάζεται *Null* και είναι μια ειδική τιμή που χρησιμοποιείται ως *ένδειξη* ότι δεν υπάρχουν άλλα στοιχεία.



Υλοποίηση συνδεδεμένης λίστας

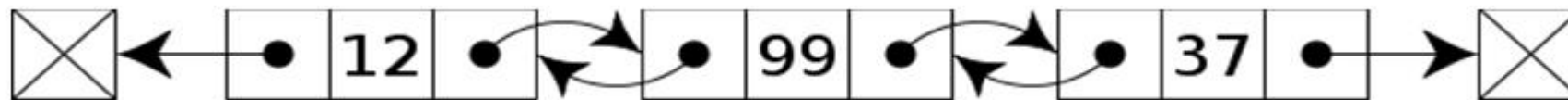
Απλά Συνδεδεμένες Λίστες

Κάθε στοιχείο περιλαμβάνει και ένα ακόμα πεδίο (συνήθως αναφέρεται ως πεδίο *Next*) το οποίο περιέχει την *αρχική διεύθυνση της μνήμης* του *επόμενου* στοιχείου της λίστας.



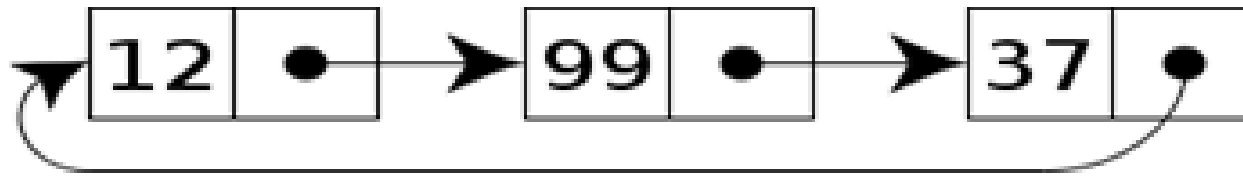
Διπλά Συνδεδεμένες Λίστες

Κάθε στοιχείο περιλαμβάνει και δυο ακόμα πεδία (συνήθως αναφέρονται ως πεδία *Prev* και *Next*) τα οποία περιέχουν αντίστοιχα τις *αρχικές διευθύνσεις μνήμης* του *προηγούμενου* και *επόμενου* στοιχείου της λίστας.



Κυκλικά συνδεδεμένες λίστες

- Απλά ή διπλά συνδεδεμένες λίστες, στις οποίες ο δείκτης *Next* του τελευταίου στοιχείου αντί για *NULL*, δείχνει πίσω στο πρώτο στοιχείο.

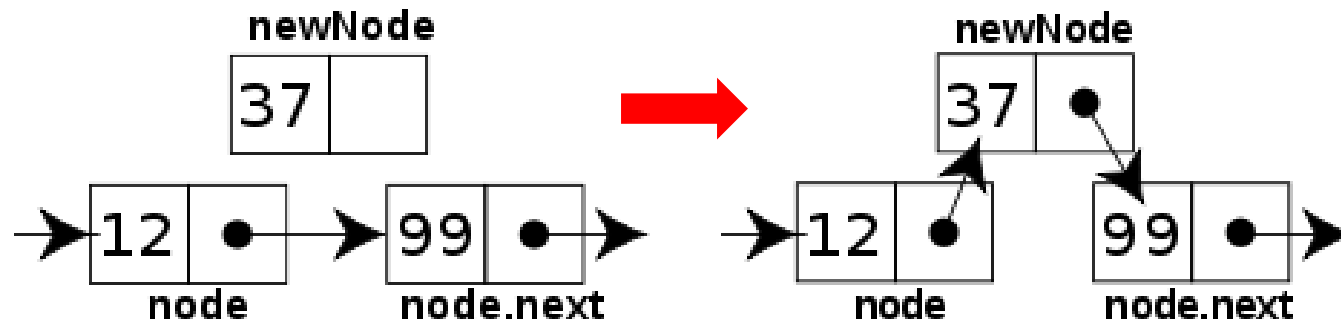


- Στις διπλά συνδεδεμένες λίστες, επιπλέον ο δείκτης *Prev* του πρώτου στοιχείου αντί για *NULL*, δείχνει στο πρώτο στοιχείο
- Σε αντιδιαστολή με τις κυκλικές, οι απλές λίστες μερικές φορές ονομάζονται και *γραμμικές*

Εισαγωγή νέου στοιχείου σε συνδεδεμένη λίστα

Γίνεται 'σπάζοντας' την λίστα σε αυτό το σημείο και εισάγοντας/προσαρμόζοντας κατάλληλα τα πεδία σύνδεσης του νέου στοιχείου με εκείνα των στοιχείων πριν και μετά από αυτό.

Κατ' αναλογία σκεφτείτε το σπάσιμο μιας αλυσίδας *ανθρώπων που κρατούνται χέρι-χέρι* ώστε να *συμπεριλάβουν* έναν ακόμα άνθρωπο.

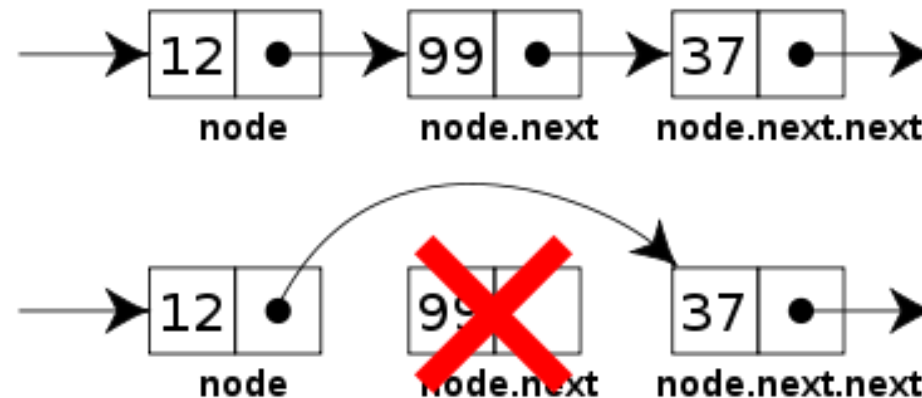


Για διπλά συνδεδεμένες λίστες η διαδικασία εισαγωγής είναι ανάλογη.

Διαγραφή στοιχείου από συνδεδεμένη λίστα

Γίνεται 'σπάζοντας' την λίστα σε αυτό το σημείο και προσαρμόζοντας κατάλληλα τα πεδία σύνδεσης των στοιχείων πριν και μετά το στοιχείο που διαγράφεται.

Κατ' αναλογία σκεφτείτε το σπάσιμο μιας αλυσίδας *ανθρώπων που κρατούνται χέρι-χέρι* ώστε να *αποχωρήσει* ένας άνθρωπος.



Για διπλά συνδεδεμένες λίστες η διαδικασία διαγραφής είναι ανάλογη.

Λεξικά (dictionaries)

Τα συναντούμε και με διάφορα εναλλακτικά ονόματα όπως *απεικονίσεις* (*maps*) *συστοιχίες συσχετισμού* (*associative arrays*), *πίνακες κατακερματισμού* (*hash tables*) κ.α

- Ένα λεξικό είναι μια δομή δεδομένων για την αποθήκευση μιας ομάδας αντικειμένων για την πρόσβαση των οποίων χρησιμοποιούμε *κλειδιά* αντί για *δείκτες*
- Ένα λεξικό αποτελείται από ένα *σύνολο κλειδιών* και κάθε *κλειδί* έχει μία μόνο *συσχετισμένη τιμή* (αντικείμενο)
- Δίνοντας ένα *κλειδί*, το λεξικό επιστρέφει *τη τιμή (ή το αντικείμενο)* που αντιστοιχεί σ' αυτό
- Αν και *οποιοσδήποτε απλός τύπος δεδομένων* μπορεί να αποτελέσει κλειδί ενός λεξικού, τα *κλειδιά* είναι συνήθως *συμβολοσειρές*

Παραδείγματα λεξικών σε διάφορες γλώσσες προγραμματισμού

| Python | Java |
|--|--|
| <pre>phonebook = { 'Sally Smart': '555-9999', 'John Doe': '555-1212', 'J. Random Hacker': '553-1337', }</pre> | <pre>var myObject = { "Sally Smart" : "555-9999", "John Doe" : "555-1212", "J. Random Hacker" : "553-1337" };</pre> |
| C++ | Lisp |
| <pre>int main() { std::map<std::string, std::string> phone_book; phone_book.insert(std::make_pair("Sally Smart", "555-9999")); phone_book.insert(std::make_pair("John Doe", "555-1212")); phone_book.insert(std::make_pair("J. Rand Hker", "553-1337")); }</pre> | <pre>'(("Sally Smart" . "555-9999") ("John Doe" . "555-1212") ("J. Random Hacker" . "553-1337"))</pre> |
| Optim J (extension of Java) | PHP |
| <pre>String[String] phoneBook = { "Sally Smart" -> "555-9999", "John Doe" -> "555-1212", "J. Random Hacker" -> "553-1337" };</pre> | <pre>\$phonebook = array(); \$phonebook['Sally Smart'] = '555-9999'; \$phonebook['John Doe'] = '555-1212'; \$phonebook['J. Random Hacker'] = '555-1337';</pre> |

Βασικές πράξεις πάνω στα λεξικά

- *Δημιουργία* λεξικού
 - Δημιουργία *κενού* λεξικού
 - Δημιουργία με *ταυτόχρονη αρχικοποίηση* (εισαγωγή) των στοιχείων του
- *Εισαγωγή στοιχείων* με τη μορφή ζευγών (κλειδί, τιμή)
 - Αν το κλειδί *δεν υπάρχει*, τότε το στοιχείο *εισάγεται*
 - Αν το κλειδί *υπάρχει*, τότε η παρούσα τιμή του *αντικαθίσταται*
- *Εύρεση τιμής* που αντιστοιχεί σε συγκεκριμένο κλειδί
 - Αν το κλειδί υπάρχει, επιστρέφεται η τιμή του
- *Διαγραφή* λεξικού