ΚΕΦΑΛΑΙΟ 9

ΘΕΜΕΛΙΩΔΕΙΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Στο Κεφάλαιο αυτό θα περιγράψουμε τους τρόπος με τους οποίους αποθηκεύουμε και διαχειριζόμαστε συλλογές από δεδομένα. Μία συλλογή δεδομένων, μαζί με τον τρόπο ή τους τρόπους με τους οποίους διαχειριζόμαστε αυτά τα δεδομένα, είναι μια δομή δεδομένων.

Πιο συγκεκριμένα, θα αναφερθούμε αρχικά στους πίνακες και στη συνέχεια στις συνδεδεμένες λίστες, καθώς πρόκειται για τους δύο πιο συχνές δομές δεδομένων που χρησιμοποιούνται στην πράξη. Όπως και στο προηγούμενο κεφάλαιο, θα γίνει προσπάθεια να οριστούν και να χρησιμοποιηθούν αυτές οι δομές μέσα από την επίλυση συγκεκριμένων παραδειγμάτων, έτσι ώστε να αναδειχθεί η χρησιμότητά τους, οι τρόποι λειτουργίας τους αλλά και οι διαφορές τους.

Σε αρκετές περιπτώσεις, μαζί με τον αλγόριθμο σε ψευδογλώσσα, θα δίδεται και το αντίστοιχο πρόγραμμα σε C, για τους αναγνώστες που ενδιαφέρονται να ασχοληθούν σε μεγαλύτερο βάθος.

9.1 Αναγκαιότητες – αρχική συζήτηση

Ας ξεκινήσουμε θεωρώντας το ακόλουθο πρόβλημα:

Πρόβλημα:

«Να γίνει αλγόριθμος που θα διαβάζει ένα γνωστό πλήθος από ακέραιους αριθμούς και θα βρίσκει το μέσο όρο τους. Στη συνέχεια να βρεθεί η απόσταση (σε ακέραιες τιμές) του κάθε αριθμού από τον υπολογισμένο μέσο όρο»

Συζήτηση:

Έστω ότι το πλήθος των αριθμών είναι 4 και έστω ότι οι αριθμοί είναι οι 5, 10, 15, 20. Προφανώς, ο μέσος όρος τους είναι 12.5 και οι αποστάσεις των αριθμών από το μέσο όρο είναι οι -7.5, -2.5, 2.5 και 7.5 αντίστοιχα. Οι αποστάσεις υπολογίστηκαν αφαιρώντας κάθε αριθμό από το μέσο όρο.

Ο υπολογισμός του μέσου όρου είναι μια απλή υπόθεση, όπως φαίνεται στο τμήμα αλγορίθμου με όνομα **Δοκιμή** που φαίνεται παραπλεύρως. Στο σημείο που σημειώνεται με το αντίστοιχο σχόλιο, ο μέσος όρος έχει υπολογιστεί. Αρκεί τώρα να προστεθεί μία ακόμη επανάληψη τύπου ΓΙΑ…, στην οποία θα εκτελείται μία αφαίρεση (του κάθε αριθμού από τον μέσο όρο) και μία εντολή εξόδου. Ωστόσο στο σημείο αυτό του αλγορίθμου η μοναδική τιμή που υπάρχει για τη μεταβλητή είναι η τελευταία που εισήχθηκε. Όλες οι άλλες τιμές έχουν χαθεί, καθώς η μεταβλητή x μπορεί να αποθηκεύει μία μόνο τιμή στη θέση μνήμης που της αντιστοιχεί. Αυτό σημαίνει ότι ο αλγόριθμος που αναπτύξαμε δεν είναι δυνατόν να λύσει το πρόβλημα.

```
ΑΛΓΟΡΙΘΜΟΣ Δοκιμή
ΜΕΤΑΒΛΗΤΕΣ
        x, i, plithos, S: ΑΚΕΡΑΙΕΣ
        ΜΟ:ΠΡΑΓΜΑΤΙΚΕΣ
ΑΡΧΗ
    ΕΠΑΝΑΛΑΒΕ
        ΔΙΑΒΑΣΕ plithos
    ΜΕΧΡΙ plithos>0
    S←0
    ΓΙΑ i←1 ΜΕΧΡΙ plithos ΑΡΧΗ
        ΔΙΑΒΑΣΕ x
        S←S+x
    ΤΕΛΟΣ
    ΜΟ←S/plithos
//σε αυτό το σημείο ο μέσος όρος έχει υπολογιστεί
ΤΕΛΟΣ
```

Εάν οι τιμές εισόδου ήταν πολύ λίγες, τότε θα ήταν δυνατόν να χρησιμοποιήσουμε μία μεταβλητή για κάθε τιμή εισόδου, ωστόσο η στρατηγική αυτή δεν λύνει το γενικό πρόβλημα όπου το πλήθος των τιμών μπορεί να είναι αυθαίρετα μεγάλο. Είναι προφανές ότι για τη λύση αυτού του, απλού φαινομενικά, προβλήματος απαιτείται ένας τρόπος αποθήκευσης πολλών ομοειδών τιμών σε μία ενιαία κατασκευή. Αυτές οι κατασκευές ονομάζονται **δομές δεδομένων**.

9.1.1 Ορισμοί

Μία δομή δεδομένων είναι μια οργανωμένη συλλογή από ομοειδή αποθηκευμένα δεδομένα. Στα δεδομένα αυτά μπορούν να εφαρμοστούν διάφορες λειτουργίες οι οποίες θα τα επεξεργάζονται.

Κάθε ένα από τα δεδομένα αποθηκεύεται σε έναν **κόμβο** της δομής δεδομένων. Ο κόμβος μπορεί να θεωρηθεί ως το μικρότερο συστατικό στοιχείο της δομής δεδομένων.

Η πιο βασική από τις λειτουργίες που εφαρμόζονται σε μια δομή δεδομένων είναι η **προσπέλαση**, ο τρόπος δηλαδή με τον οποίο μπορούμε να εισάγουμε, να

εμφανίσουμε ή να αλλάξουμε τα δεδομένα σε έναν κόμβο. Κατά συνέπεια, θα πρέπει να υπάρχει ένας μονοσήμαντος τρόπος να προσπελαύνονται τα δεδομένα μιας δομής δεδομένων με τη χρήση ενός ονόματος που αντιστοιχίζεται στη δομή.

Η **εισαγωγή** ενός νέου κόμβου είναι η προσθήκη του σε μια ήδη υπάρχουσα δομή δεδομένων. Η εισαγωγή δεν πρέπει να συγχέεται με την προσπέλαση.

Η **διαγραφή** ενός κόμβου είναι η απομάκρυνσή του από την υπάρχουσα δομή. Η διαγραφή ενός κόμβου έχει ως αποτέλεσμα να μην είναι προσπελάσιμα τα δεδομένα που έχουν αποθηκευτεί σε αυτό τον κόμβο και δεν σχετίζεται με τον μηδενισμό των δεδομένων.

Η **αναζήτηση** σε μια δομή δεδομένων αφορά στον έλεγχο της ύπαρξης ή μη ύπαρξης μια συγκεκριμένης τιμής σε έναν από του κόμβους της δομής δεδομένων.

Η **ταξινόμηση** μιας δομής δεδομένων είναι η αναδιάταξη των κόμβων έτσι ώστε οι αποθηκευμένες τιμές να ικανοποιούν μια δεδομένη σχέση διάταξης.

Η **συγχώνευση** δύο δομών δεδομένων αφορά στη συνένωση δύο δομών σε μία. Κατά τη συγχώνευση είναι πιθανόν να τίθενται και επιπλέον περιορισμοί που να σχετίζονται με την τελική οργάνωση της νέας δομής.

Η **αντιγραφή** μιας δομής δεδομένων σε μια άλλη περιλαμβάνει την αντιγραφή κάποιων ή όλων των δεδομένων μιας δομής σε μία δεύτερη δομή. Κατά την αντιγραφή αυτή μπορούν να ισχύουν κάποιοι κανόνες.

Είναι φανερό ότι για την ίδια συλλογή δεδομένων και για το ίδιο πρόβλημα μπορούν να υπάρχουν πολλές δομές δεδομένων, οι οποίες να είναι διαφορετικές μεταξύ τους. Σε κάθε περίπτωση, η επιλογή της κατάλληλης δομής δεδομένων είναι μια στρατηγική απόφαση η οποία πρέπει να ληφθεί παράλληλα με την επιλογή της αλγοριθμικής τακτικής που θα ακολουθήσουμε. Η επιλογή μιας δομής δεδομένων μπορεί να επηρεάσει ακόμη και τον αλγόριθμο που θα αναπτυχθεί, ενώ προφανώς ισχύει και το αντίστροφο. Κατά συνέπεια, το τελικό προϊόν (το πρόγραμμα) θα προκύψει από μια σειρά επιλογών και για τον αλγόριθμο αλλά και για τη δομή δεδομένων που θα προκριθούν. Η γνωστή εξίσωση (N. Wirth) έχει ουσιαστική και δομική σημασία σε όλα τα προγράμματα: *Δομές Δεδομένων + Αλγόριθμοι = Προγράμματα.*

9.1.2 Το ζήτημα του μεγέθους (πλήθους)

Η επιλογή μια κατάλληλης δομής δεδομένων επηρεάζεται σε καθοριστικό βαθμό από την εκ των προτέρων γνώση του μεγέθους της δομής, δηλαδή του πλήθους των δεδομένων που θα αποθηκευτούν στη δομή δεδομένων.

Σε πολλές περιπτώσεις είναι δυνατόν να γνωρίζουμε από την αρχή αυτό το πλήθος το οποίο μπορεί να γίνει γνωστό είτε από το ίδιο το πρόβλημα (από την εκφώνησή του), είτε από μια απλή προεπεξεργασία – μέτρηση του πλήθους - των δεδομένων. Είναι φανερό ότι σε δεδομένα τα οποία έχουν ήδη καταγραφεί (ιστορικά δεδομένα) ή τα οποία προκύπτουν από μια συγκεκριμένη και σταθερή στο χρόνο διεργασία, η μέτρηση του πλήθους τους είναι εύκολη. Για παράδειγμα, σε έναν

τηλεφωνικό κατάλογο, ο οποίος αλλάζει κάθε έτος, είναι εύκολο να μετρήσουμε το πλήθος των δεδομένων. Επίσης, μια συλλογή από αισθητήρες που μετρούν πχ τις θερμοκρασίες σε μια αποθήκη είναι οπωσδήποτε συγκεκριμένου και γνωστού πλήθους. Σε αρκετές περιπτώσεις ακόμη κι αν δεν γνωρίζουμε το ακριβές πλήθος των δεδομένων, είναι πιθανόν να γνωρίζουμε ένα άνω όριο αυτού του πλήθους. Έτσι, για παράδειγμα, το γεγονός ότι σε μία σχολική τάξη ο νόμος ορίζει ότι το μέγιστο πλήθος είναι 25 άτομα, σημαίνει ότι αυτό το πλήθος δεν μπορεί ποτέ να ξεπεραστεί, οπότε μπορούμε να εργαστούμε σε όρους συγκεκριμένου πλήθους. Στην περίπτωση βέβαια αυτή, σπάνια θα υπάρχουν τάξεις με ακριβώς 25 άτομα, κάτι που σημαίνει ότι συνήθως θα υπάρχουν κενές θέσεις σε αυτή τη δομή δεδομένων.

Η ξεκάθαρη επιλογή, όταν γνωρίζουμε το πλήθος των δεδομένων, είναι η χρήση μιας **στατικής** δομής δεδομένων. Σε μία στατική δομή ισχύουν τα ακόλουθα:

1. Το μέγεθος είναι γνωστό και σταθερό.
2. Εφόσον το μέγεθος είναι γνωστό, η απαραίτητη μνήμη όπου θα αποθηκευτεί η δομή δεσμεύεται στην αρχή του προγράμματος. Μάλιστα, όλα τα δεδομένα αποθηκεύονται σε συνεχόμενες θέσεις μνήμης
3. Η εισαγωγή και η διαγραφή δεν υφίστανται σε μια στατική δομή δεδομένων.

Όταν το πλήθος των δεδομένων που θα αποθηκευτούν είναι άγνωστο, τότε η μοναδική επιλογή που υπάρχει είναι η χρήση μιας **δυναμικής** δομής δεδομένων. Σε μια δυναμική δομή δεδομένων ισχύουν τα ακόλουθα:

1. Το μέγεθος δεν είναι γνωστό ούτε σταθερό.
2. Δεν είναι δυνατόν να δεσμευτεί η απαιτούμενη μνήμη από την αρχή. Αντίθετα, θα πρέπει να υπάρχει ένας τρόπος δέσμευσης και αποδέσμευσης τεμαχίων μνήμης κατάλληλου μεγέθους *(δυναμική παραχώρηση μνήμης – dynamic storage allocation)*, ο οποίος θα λειτουργεί κατά τη διάρκεια του προγράμματος. Όταν απαιτείται, (πχ λόγω μιας εισαγωγής ενός κόμβου) τότε θα δεσμεύεται μνήμη, ενώ όταν γίνεται μια διαγραφή κόμβου τότε η μνήμη στην οποία είχε αποθηκευτεί ο κόμβος θα πρέπει να αποδεσμευτεί και να απελευθερωθεί προς άλλη χρήση.

Είναι προφανές ότι οι δυναμικές δομές δεδομένων είναι περισσότερο ευέλικτες από τις στατικές. Μάλιστα, οποιαδήποτε στατική δομή μπορεί να αναπαρασταθεί δυναμικά, ενώ το αντίστροφο δεν ισχύει (λόγω του περιορισμού του μεγέθους). Ωστόσο, μολονότι από την έως τώρα συζήτηση φαίνεται ότι οι δυναμικές δομές δεδομένων είναι περισσότερο «χρήσιμες» ή πάντως πιο «ολοκληρωμένες» και «οικονομικές», η αλήθεια είναι ότι απαιτούν πολύ περισσότερο κόπο από την πλευρά του προγραμματιστή σε σχέση με τις στατικές δομές δεδομένων. Όπως θα φανεί σε επόμενες παραγράφους, ο σωστός προγραμματισμός με δυναμικές δομές απαιτεί ιδιαίτερη προσοχή σε πολλές οριακές περιπτώσεις οι οποίες μπορούν να θέσουν τη σταθερότητα ενός προγράμματος σε περιπέτειες. Στην πραγματικότητα, η πρώτη σκέψη που κάνει κάποιος προγραμματιστής είναι η δυνατότητα επίλυσης του

προβλήματος με τη βοήθεια στατικών δομών, ακόμα κι αν αυτό σημαίνει ότι μπορεί να θυσιάσει κάποιο ποσό μνήμης.

Τελειώνοντας, ας αναφερθεί οτι υπάρχει και η δυνατότητα δημιουργίας δομών δεδομένων μικτού τύπου, οι οποίες αποτελούνται από ένα στατικά τμήματα τα οποία μπορούν να περιέχουν δυναμικά τμήματα. Με μια τέτοια περίπτωση θα ασχοληθούμε σε μια εφαρμογή στο τέλος του Κεφαλαίου.

9.2 Στατικές δομές δεδομένων – Πίνακες

Οι πίνακες είναι οι βασικές και απλούστερες στατικές δομές δεδομένων. Πίνακες υπάρχουν σε όλες τις κλασικές γλώσσες προγραμματισμού από τις πρώτες ακόμα ιστορικά εκδόσεις τους. Πρόκειται για εξαιρετικά χρήσιμα εργαλεία, αρκετά τα οποία είναι αρκετά απλά στην υλοποίησή τους.

Ένας πίνακας είναι μια συλλογή από ομοειδή στοιχεία, συγκεκριμένου και γνωστού πλήθους, τα οποία αποθηκεύονται δίπλα-δίπλα στην κεντρική μνήμη. Κάθε πίνακας χαρακτηρίζεται από ένα όνομα. Έτσι, στο Σχήμα 9.1 φαίνεται ο πίνακας Α και ο πίνακας Β.

1	*2*	*3*	*4*	*5*	*6*
‘α’	‘ζ’	‘Ν’	‘ο’	‘σ’	‘Κ’

Πίνακας με όνομα Α

	1	*2*	*3*	*4*
1	2	4	1	-1
2	8	3	4	2
3	1	1	0	2

Πίνακας με όνομα Β

Σχήμα 9.1

Ο πίνακας *Α* περιέχει μόνο χαρακτήρες (συνολικά 6) μέσα σε έξι **θέσεις**, οργανωμένες η μία μετά την άλλη σε μία γραμμή, ενώ ο πίνακας *Β* περιέχει μόνο ακέραιους αριθμούς (συνολικά 3Χ4=12) σε δώδεκα θέσεις, οργανωμένες σε τρεις **γραμμές** και τέσσερις **στήλες**. Ο πίνακας *Α* λέγεται **μονοδιάστατος** πίνακας, ενώ ο πίνακας *Β* λέγεται **δισδιάστατος** πίνακας, λόγω της οργάνωσης που προαναφέρθηκε. Εάν τοποθετήσουμε πάνω στον πίνακα *Β* ακόμη έναν όμοιο πίνακα, τότε θα μπορούσαμε να μιλήσουμε για τρισδιάστατο πίνακα, κ.ο.κ. Ένας μονοδιάστατος πίνακας Α με Κ θέσεις, μπορεί να συμβολίζεται και ως A_K, ενώ ένας δισδιάστατος πίνακας Β με Ν γραμμές και Μ στήλες μπορεί να συμβολίζεται και ως B_{NXM}

Στο Σχήμα 9.1 χρησιμοποιούμε για λόγους ευκολίας τους αριθμούς των θέσεων με πλάγια γραφή. Σημειώνεται ότι οι αριθμοί αυτοί **δεν** ανήκουν στα δεδομένα που είναι αποθηκευμένα στους πίνακες. Παρέχουν όμως τον τρόπο με τον οποίο μπορούμε να προσπελάσουμε οποιονδήποτε πίνακα.

Εφόσον οι πίνακες είναι στατικές δομές δεδομένων, το μέγεθός τους πρέπει να είναι γνωστό κατά τη γραφή του αλγόριθμου και να δηλωθεί στο τμήμα δηλώσεων των μεταβλητών. Οι αντίστοιχες δηλώσεις για τους πίνακες *Α* και *Β* θα ήταν:

```
ΜΕΤΑΒΛΗΤΕΣ
        Α[6]: ΧΑΡΑΚΤΗΡΕΣ
        Β[3,4]:ΑΚΕΡΑΙΕΣ
.....
```

Κατά τη δήλωση ενός μονοδιάστατου πίνακα, δηλώνουμε το όνομά του, το μέγεθός του και τον τύπο των δεδομένων που αποθηκεύει. Κατά τη δήλωση ενός δισδιάστατου πίνακα, εκτός από το όνομα, το πλήθος και τον τύπο των δεδομένων, πρέπει να αναφερθούμε και στην οργάνωσή τους σε γραμμές και στήλες. Έτσι, ένας πίνακας *Γ[5,2]* αποτελείται από 5 γραμμές και 2 στήλες ενώ ένας πίνακας *Δ[2,5]* αποτελείται από 2 γραμμές και 5 στήλες. Από σύμβαση, θεωρούμε πάντα ότι ο πρώτος αριθμός αφορά στις γραμμές και ο δεύτερος αριθμός αφορά στις στήλες.

9.2.1 Προσπέλαση και βασικές εργασίες

9.2.1.1 Μονοδιάστατος πίνακας

Η προσπέλαση των στοιχείων ενός πίνακα γίνεται με τη βοήθεια του ονόματος και της θέσης των στοιχείων. Έτσι, για να προσπελάσουμε το 4^{ο} στοιχείο του πίνακα *A*, το οποίο είναι το 'ο', αρκεί να γράψουμε *A[4]*. Κατά συνέπεια, μια εντολή όπως η **ΓΡΑΨΕ Α[2]** θα είχε ως αποτέλεσμα την εμφάνιση στην οθόνη του χαρακτήρα 'ζ'. Ομοίως, μια εντολή όπως η **ΔΙΑΒΑΣΕ Α[3]**, θα ανέμενε την εισαγωγή ενός χαρακτήρα από το πληκτρολόγιο. Εάν υποθέσουμε ότι πατήθηκε ο χαρακτήρας 'χ' και το <enter>, τότε το 'χ' θα αποθηκευόταν στην 3^{η} θέση του πίνακα *A*, αντικαθιστώντας βέβαια την τιμή που προϋπάρχει (το 'Ν'). Ο αριθμός της θέσης ενός στοιχείου, ονομάζεται και δείκτης αυτής της θέσης.

Έχοντας τα παραπάνω υπόψη, ο αλγόριθμος του Σχήματος 9.2 «γεμίζει» από το πληκτρολόγιο και εμφανίζει στην οθόνη έναν μονοδιάστατο πίνακα Α, 10 ακεραίων θέσεων.

```
ΑΛΓΟΡΙΘΜΟΣ Εισ_Εξ
ΜΕΤΑΒΛΗΤΕΣ
        i, A[10]: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ i←1 ΜΕΧΡΙ 10 ΑΡΧΗ
        ΔΙΑΒΑΣΕ A[i]
    ΤΕΛΟΣ
    //……..
    ΓΙΑ i←1 ΜΕΧΡΙ 10 ΑΡΧΗ
        ΓΡΑΨΕ A[i]
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int i, A[11]; //οι πίνακες ξεκινούν από το 0 στη C
   for(i=1;i<=10;i++){
      scanf("%d", &A[i]);  fflush(stdin);
   }
//……..
   for(i=1;i<=10;i++){
      printf("%d ", A[i]);
   }
}
```

Σχήμα 9.2

Όπως φαίνεται στο Σχήμα 9.2, χρησιμοποιούμε το μετρητή μιας επανάληψης **ΓΙΑ…** ως δείκτη των θέσεων του πίνακα.

Σημαντικό είναι να γίνει πλήρως αντιληπτό ότι μετά την εισαγωγή όλων των στοιχείων του πίνακα Α (μετά δηλαδή το τέλος της πρώτης επανάληψης), τα στοιχεία αυτά βρίσκονται αποθηκευμένα στον πίνακα. Έτσι, στο σημείο όπου βρίσκονται τα σχόλια, μπορούμε να χρησιμοποιήσουμε αυτά τα στοιχεία προσπελαύνοντάς τα. Εφόσον δεν αλλάξουμε τις τιμές αυτές, η δεύτερη επανάληψη θα τις εμφανίσει στην οθόνη.

Έχοντας υπόψη αυτή την ιδιότητα του πίνακα (να αποθηκεύει δηλαδή τις τιμές του για όλη τη διάρκεια του προγράμματος), μπορούμε να επιστρέψουμε στο αρχικό μας πρόβλημα (παράγραφος 9.1), ελαφρώς παραλλαγμένο.
Ας υποθέσουμε ότι το πλήθος των αριθμών είναι γνωστό πριν τον προγραμματισμό (έστω ότι είναι 10 στοιχεία). Τότε μπορούμε να λύσουμε το πρόβλημα χρησιμοποιώντας έναν μονοδιάστατο πίνακα 10 στοιχείων ως εξής:

```
ΑΛΓΟΡΙΘΜΟΣ Δοκιμή_1
ΜΕΤΑΒΛΗΤΕΣ
        X[10], i, S: ΑΚΕΡΑΙΕΣ
        ΜΟ:ΠΡΑΓΜΑΤΙΚΕΣ
ΑΡΧΗ
    S←0
    ΓΙΑ i←1 ΜΕΧΡΙ 10 ΑΡΧΗ
        ΔΙΑΒΑΣΕ x[i]
        S←S+x[i]
    ΤΕΛΟΣ
    ΜΟ←S/10
    ΓΙΑ i←1 ΜΕΧΡΙ 10 ΑΡΧΗ
        ΓΡΑΨΕ ΜΟ - x[i]
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

Στην πρώτη επανάληψη διαβάζουμε ολόκληρο τον πίνακα, υπολογίζοντας ταυτόχρονα και το άθροισμα των στοιχείων του, ενώ στη δεύτερη επανάληψη, αφαιρούμε κάθε στοιχείο από τον ήδη υπολογισμένο μέσο όρο.

Είναι βέβαια φανερό οτι εάν το πλήθος των στοιχείων ήταν άγνωστο ή, έστω, δινόταν από το πληκτρολόγιο, δεν θα ήταν δυνατόν να χρησιμοποιήσουμε πίνακα, καθώς δεν θα είχαμε καμία ένδειξη για το μέγεθός του, οπότε θα ήταν αδύνατον να τον δηλώσουμε.

9.2.1.2 Δισδιάστατος πίνακας

Η θέση κάθε στοιχείου ενός δισδιάστατου πίνακα καθορίζεται όχι μόνο από έναν αλλά από δύο δείκτες. Ο πρώτος δείκτης αφορά στη γραμμή και ο δεύτερος στη στήλη του πίνακα. Έτσι, στον πίνακα *B* που ακολουθεί, το στοιχείο *B[1,2]* είναι το 4, ενώ το στοιχείο *B[3,3]* είναι το 0.

	1	*2*	*3*	*4*
1	2	4	1	-1
2	8	3	9	2
3	1	1	0	2

Πίνακας Β

Για να «γεμίσουμε» έναν δισδιάστατο πίνακα, θα πρέπει πρώτα να αποφασίσουμε εάν θα εργαστούμε «κατά γραμμή» ή «κατά στήλη».
Στην πρώτη περίπτωση, θα πρέπει να εισαχθούν κατά σειρά τα στοιχεία [1,1], [1,2], [1,3], [1,4], [2,1], [2,2], [2,3], [2,4], [3,1], [3,2], [3,3], [3,4], θα προσπελάσουμε δηλαδή πρώτα την $1^η$ γραμμή, ύστερα τη $2^η$ γραμμή κ.ο.κ.

Στη δεύτερη περίπτωση θα πρέπει να εισαχθούν κατά σειρά τα [1,1], [2,1], [3,1], [2,1], [2,2], [2,3], [3,1], [3,2], [3,3], [4,1], [4,2], [4,3], δηλαδή πρώτα τα στοιχεία της $1^{ης}$ στήλης ύστερα της $2^{ης}$ στήλης κ.ο.κ.
Στο Σχήμα 9.3 που ακολουθεί χρησιμοποιούμε και τους δύο τρόπους για την εισαγωγή και εμφάνιση ενός δισδιάστατου πίνακα 10Χ5 (δέκα γραμμών και πέντε στηλών).

```
ΑΛΓΟΡΙΘΜΟΣ Εισ_Εξ_κατά_Γραμμή
ΜΕΤΑΒΛΗΤΕΣ
        i, j, A[10,5]: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ i←1 ΜΕΧΡΙ 10 ΑΡΧΗ
        ΓΙΑ j←1 ΜΕΧΡΙ 5 ΑΡΧΗ
            ΔΙΑΒΑΣΕ A[i, j]
        ΤΕΛΟΣ
    ΤΕΛΟΣ
    //……..
    ΓΙΑ i←1 ΜΕΧΡΙ 10 ΑΡΧΗ
        ΓΙΑ j←1 ΜΕΧΡΙ 5 ΑΡΧΗ
            ΓΡΑΨΕ A[i, j]
        ΤΕΛΟΣ
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int i, j, A[11][6]; //οι πίνακες ξεκινούν από το 0 στη C
   for(i=1;i<=10;i++){
      for(j=1;j<=5;j++){
         scanf("%d", &A[i][j]);  fflush(stdin);
      }
   }
//……..
   for(i=1;i<=10;i++){
      for(j=1;j<=5;j++){
         printf("%d ", A[i][j]);
      }
   }getchar();
}
```

```
ΑΛΓΟΡΙΘΜΟΣ Εισ_Εξ_κατά_Στήλη
ΜΕΤΑΒΛΗΤΕΣ
        i, j, A[10,5]: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ j←1 ΜΕΧΡΙ 5 ΑΡΧΗ
        ΓΙΑ i←1 ΜΕΧΡΙ 10 ΑΡΧΗ
            ΔΙΑΒΑΣΕ A[i, j]
        ΤΕΛΟΣ
    ΤΕΛΟΣ
    //……..
    ΓΙΑ j←1 ΜΕΧΡΙ 5 ΑΡΧΗ
        ΓΙΑ i←1 ΜΕΧΡΙ 10 ΑΡΧΗ
            ΓΡΑΨΕ A[i, j]
        ΤΕΛΟΣ
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int i, j, A[11][6]; //οι πίνακες ξεκινούν από το 0 στη C
   for(j=1;j<=5;j++){
      for(i=1;i<=10;i++){
         scanf("%d", &A[i][j]);  fflush(stdin);
      }
   }
//……..
   for(j=1;j<=5;j++){
      for(i=1;i<=5;i++){
         printf("%d ", A[i][j]);
      }
   }getchar();
}
```

Σχήμα 9.3

Η χρήση των εμφωλευμένων επαναλήψεων μπορεί να χρησιμοποιηθεί σε κάθε περίπτωση προσπέλασης και επεξεργασίας των δισδιάστατων πινάκων και όχι μόνο στην είσοδο και έξοδο δεδομένων.

9.2.1.3 Ειδικές περιπτώσεις δισδιάστατων πινάκων

- Όταν το πλήθος των γραμμών είναι ίσο με το πλήθος των στηλών ενός δισδιάστατου πίνακα, τότε ο πίνακας αυτός λέγεται **τετραγωνικός**. Για παράδειγμα, ένας πίνακας $A_{κΧκ}$ είναι τετραγωνικός καθώς έχει κ γραμμές και κ στήλες (κ γνωστό).
- Σε έναν τετραγωνικό πίνακα, τα στοιχεία που έχουν τον ίδιο δείκτη γραμμής και στήλης, είναι δηλαδή της μορφής *A[i,i], i=1, 2, ...κ*, ανήκουν στην **κύρια διαγώνιο**. Έτσι, στον πίνακα $A_{κΧκ}$ τα στοιχεία *A[1,1], A[2,2], A[3,3]*, κλπ ανήκουν στην κύρια διαγώνιο του πίνακα.

- Σε έναν τετραγωνικό πίνακα $A_{\kappa X\kappa}$, τα στοιχεία *A[i,k-i+1],* ανήκουν στη **δευτερεύουσα διαγώνιο** του πίνακα.
- Σε έναν τετραγωνικό πίνακα $A_{\kappa X\kappa}$, όταν *A[i, j]=A[j, i],* για κάθε *i,j=1,2,...κ*, τότε ο πίνακας είναι **συμμετρικός** ως προς την κύρια διαγώνιό του.
- Ένας τετραγωνικός πίνακας που περιέχει **μηδενικά** (0) σε όλα τα στοιχεία *A[i,j], για i>j*, λέγεται **άνω τριγωνικός**.
- Αντίστοιχα, ένας **κάτω τριγωνικός** πίνακας περιέχει μηδενικά (0) σε όλα τα στοιχεία *A[i,j], i<j*.
- Ένας τετραγωνικός πίνακας $A_{\kappa X\kappa}$ λέγεται κεντροσυμμετρικός, όταν όλα τα στοιχεία του είναι συμμετρικά ίσα ως προς το κέντρο του πίνακα. Στους κεντροσυμμετρικούς πίνακες ισχύει: A[i, j] =A[κ-i+1,κ-j+1].

Στο Σχήμα 9.4 που ακολουθεί φαίνονται όλες οι παραπάνω περιπτώσεις.

3	2	4	5
1	3	9	1
1	2	0	2
7	4	3	2

Τετραγωνικός πίνακας

3	2	4	5
1	**3**	9	1
1	2	**0**	2
7	4	3	**2**

Στοιχεία της κυρίας διαγωνίου

3	2	4	**5**
1	3	**9**	1
1	**2**	0	2
7	4	3	2

Στοιχεία της δευτερεύουσας διαγωνίου

3	2	4	5
2	3	9	1
4	9	0	3
5	1	3	2

Συμμετρικός πίνακας

3	2	4	5
0	3	9	1
0	**0**	1	3
0	**0**	**0**	2

Άνω τριγωνικός πίνακας

3	**0**	**0**	**0**
2	3	**0**	**0**
4	9	1	**0**
5	1	3	2

Κάτω τριγωνικός πίνακας

2	3	4	6
5	7	0	9
9	0	7	5
6	4	8	2

Κεντροσυμμετρικός πίνακας με άρτιο πλήθος γραμμών και στηλών
(το κέντρο είναι στην τομή των διπλών γραμών)

2	8	4	6	5
7	0	9	10	1
3	5	12	5	3
1	10	9	0	7
5	6	4	8	2

Κεντροσυμμετρικός πίνακας με περιττό πλήθος γραμμών και στηλών
(το κέντρο είναι το στοιχείο [3,3])

Σχήμα 9.4

Όλες οι παραπάνω ειδικές περιπτώσεις δισδιάστατων πινάκων έχουν ιδιαίτερη σημασία σε ειδικές εφαρμογές ή στην αναπαράσταση άλλων, περισσότερο περίπλοκων δομών δεδομένων.

9.2.2 Βασικές εφαρμογές στους πίνακες

Εκτός από τις βασικές λειτουργίες επί των δομών δεδομένων, κυρίως την αναζήτηση, την ταξινόμηση και τη συγχώνευση, οι οποίες ήδη αναφέρθηκαν, και στις οποίες θα επανέρθουμε σε επόμενες παραγράφους και κεφάλαια, οι πίνακες (μονοδιάστατοι και δισδιάστατοι) χρησιμοποιούνται σε πολλές και διαφορετικές περιπτώσεις πραγματικών εφαρμογών. Στις περισσότερες από αυτές τις εφαρμογές χρησιμοποιούνται κάποιοι τυποποιημένοι αλγόριθμοι, με κάποιους από τους οποίους θα ασχοληθούμε στη συνέχεια με τη μορφή λυμένων προβλημάτων. Ο τρόπος της μελέτης μέσω εφαρμογών κρίνεται ως ιδιαίτερα χρήσιμος για την πλήρη κατανόηση της λειτουργίας και της χρησιμότητας αυτών των δομών δεδομένων.

9.2.2.1 Εύρεση αθροίσματος μονοδιάστατου πίνακα

Πρόβλημα:
«Έστω μονοδιάστατος πίνακας με 10 ακέραιους αριθμούς. Να βρεθεί το άθροισμα των θετικών και ο μέσος όρος των αρνητικών αριθμών»

Συζήτηση:
Ήδη έχει συζητηθεί η περίπτωση του μέσου όρου όλων των στοιχείων ενός μονοδιάστατου πίνακα. Η επίλυση του συγκεκριμένου προβλήματος δεν παρουσιάζει αλγοριθμικά καμία διαφορά από τη λύση που δόθηκε για το ίδιο πρόβλημα σε προηγούμενο Κεφάλαιο.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΥΠΟΛΟΓΙΣΜΟΙ
ΜΕΤΑΒΛΗΤΕΣ
    Χ[10], Σ_Θ, Σ_Α, ΠΛΗΘΟΣ_Α, i: ΑΚΕΡΑΙΕΣ
    Μ_Ο_Α: ΠΡΑΓΜΑΤΙΚΕΣ
ΑΡΧΗ
    Σ_Θ←0                   //αρχικοποίηση αθροίσματος θετικών
    Σ_Α←0                   //αρχικοποίηση αθροίσματος αρνητικών
    ΠΛΗΘΟΣ_Α←0              //αρχικοποίηση πλήθους αρνητικών
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
        ΔΙΑΒΑΣΕ Χ[i]
        ΑΝ Χ[i]>0 ΤΟΤΕ
                Σ_Θ ← Σ_Θ +Χ
        ΑΛΛΙΩΣ
                ΑΝ Χ[i]<0 ΤΟΤΕ ΑΡΧΗ
                        Σ_Α← Σ_Α+Χ[i]
                        ΠΛΗΘΟΣ_Α ←ΠΛΗΘΟΣ_Α+1
                ΤΕΛΟΣ

    ΤΕΛΟΣ                   //επανάληψης και εσωτερικών υπολογισμών
    ΓΡΑΨΕ Σ_Θ
    ΑΝ ΠΛΗΘΟΣ_Α<>0 ΤΟΤΕ ΑΡΧΗ
        Μ_Ο_Α ←Σ_Α/ΠΛΗΘΟΣ_Α
        ΓΡΑΨΕ Μ_Ο_Α
    ΤΕΛΟΣ
```

ΤΕΛΟΣ

9.2.2.2 Μέγιστα και ελάχιστα σε μονοδιάστατους πίνακες.

Πρόβλημα:

«Δίνεται μονοδιάστατος πίνακας 10 ακεραίων. Να βρεθεί η μέγιστη τιμή, η ελάχιστη τιμή και οι θέσεις στις οποίες υπάρχουν αυτά τα μέγιστα και ελάχιστα.»

Συζήτηση:

Αλγοριθμικά, η εύρεση της μέγιστης και ελάχιστης τιμής δεν παρουσιάζει κάποια δυσκολία. Ενδιαφέρον στην περίπτωση αυτή έχει το δεύτερο σκέλος του προβλήματος, το οποίο αφορά στις θέσεις όπου βρέθηκε η μέγιστη ή η ελάχιστη τιμή. Η λύση αυτού του ζητήματος απαιτεί οπωσδήποτε την αποθήκευση των στοιχείων σε πίνακα, κατ' απόλυτη αντιστοιχία με το πρώτο παράδειγμα αυτού του κεφαλαίου.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΜΕΓ_ΕΛΑΧ
ΜΕΤΑΒΛΗΤΕΣ
     Χ[10], max, min, i: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
          ΔΙΑΒΑΣΕ Χ[i]
    ΤΕΛΟΣ
    max←Χ[1]
    min←Χ[1] //αρχικοποίηση στο πρώτο στοιχείο
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
          ΑΝ Χ[i]>max ΤΟΤΕ
                max←Χ[i]
          ΑΝ Χ[i[<min ΤΟΤΕ
                min←Χ[i]
    ΤΕΛΟΣ
//σε αυτό το σημείο, η μέγιστη και η ελάχιστη τιμή έχουν
υπολογιστεί
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ   //διαπερνούμε από την
αρχή τον πίνακα
          ΑΝ Χ[i]=max ΤΟΤΕ
                ΓΡΑΨΕ 'βρέθηκε μέγιστος στη θέση ', i
          ΑΝ Χ[i[=min ΤΟΤΕ
                ΓΡΑΨΕ 'βρέθηκε ελάχιστος στη θέση ', i
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int x[11], i, max, min;
   for(i=1;i<=10;i++){
      scanf("%d", &x[i]);fflush(stdin);
      }
   max=x[1];min=x[1];
   for(i=1;i<=10;i++){
      if(x[i]>max)
        max=x[i];
      if (x[i]<min)
        min=x[i];
   }
   for(i=1;i<=10;i++){
      if(x[i]==max)
        printf("max in position %d\n", i);
      if (x[i]==min)
        printf("min in position %d\n", i);;
   }
   getchar();
}
```

9.2.2.3 Απόφαση σε μονοδιάστατο πίνακα (1)

Πρόβλημα:

«Δίνεται μονοδιάστατος πίνακας 10 ακεραίων. Να βρεθεί εάν τα στοιχεία του είναι όλα θετικά ή όχι».

Συζήτηση:

Τα προβλήματα στα οποία η λύση δεν είναι αριθμητική αλλά είναι μια απάντηση τύπου **ΝΑΙ** ή **ΟΧΙ** λέγονται **προβλήματα απόφασης**. Όλα τα προβλήματα απόφασης λύνονται με παρόμοιο τρόπο. Το πρόβλημα που συζητούμε είναι ένα τυπικό πρόβλημα αυτής της κατηγορίας και μπορεί εύκολα να παραλλαγεί, αλλάζοντας τον

τελικό έλεγχο. Έτσι, θα μπορούσαμε να ελέγξουμε εάν όλα τα στοιχεία του πίνακα είναι άρτια ή αν όλα είναι περιττά ή όλα μη μηδενικά κλπ.
Για τη λύση αυτών των προβλημάτων, χρησιμοποιούμε μία λογική μεταβλητή η οποία περιέχει ουσιαστικά την απόφασή μας. Η μεταβλητή αυτή αρχικοποιείται στην τιμή **true**. Η σημασία αυτής της αρχικοποίησης είναι ότι θεωρούμε ότι η πρόταση ισχύει. Στη συνέχεια, διαπερνούμε όλα τα στοιχεία του πίνακα και αναζητούμε έστω και μία περίπτωση η οποία να μην ικανοποιεί την υπόθεσή μας. Αν υπάρχει ένα τέτοιο στοιχείο, η λογική μεταβλητή αλλάζει τιμή και γίνεται false. Η διαπέραση μπορεί να γίνει σε ολόκληρο τον πίνακα (με μια εντολή **ΓΙΑ**…) ή να σταματήσει εάν βρεθεί ένα στοιχείο που να αλλάζει την τιμή της λογικής μεταβλητής (με μια εντολή **ΟΣΟ…**).

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΘΕΤΙΚΟΙ_ΑΡΙΘΜΟΙ
ΜΕΤΑΒΛΗΤΕΣ
     Χ[10], i: ΑΚΕΡΑΙΕΣ
     θετικοι: ΛΟΓΙΚΕΣ
ΑΡΧΗ
     ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
          ΔΙΑΒΑΣΕ Χ[i]
     ΤΕΛΟΣ
     θετικοί← true //έστω ότι όλοι αριθμοί είναι θετικοί
     i←1
     ΟΣΟ ((i<=10) and (θετικοί = true)) ΑΡΧΗ
          AN Χ[i]<0 ΤΟΤΕ
               θετικοί←false   //βρέθηκε ένας αρνητικός
αριθμός
          i←i+1
     ΤΕΛΟΣ
//μετά το τέλος της επανάληψης η μεταβλητή θετικοί,
περιέχει το αποτέλεσμα
     AN θετικοί = true  ΤΟΤΕ
          ΓΡΑΨΕ 'όλοι οι αριθμοί είναι θετικοί'
     ΑΛΛΙΩΣ
          ΓΡΑΨΕ 'ΔΕΝ είναι όλοι οι αριθμοί θετικοί'
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int x[11], i, positive;
   for(i=1;i<=10;i++){
      scanf("%d", &x[i]);fflush(stdin);
      }
   positive=1;
   i=1;
   while((i<=10)&&(positive)){
      if(x[i]<0)
        positive=0;
      i++;
   }
   if(positive)
     printf("all are positives");
   else
     printf("at least one negative");
   getchar();
}
```

9.2.2.4 Απόφαση σε μονοδιάστατο πίνακα (2)

Πρόβλημα:
«Έστω ένα πίνακας 10 χαρακτήρων. Μέσα στον πίνακα αποθηκεύεται ένα αλφαριθμητικό ίδιου μεγέθους. Να βρεθεί εάν το αλφαριθμητικό είναι καρκινικό (παλίνδρομο), διαβάζεται δηλαδή το ίδιο με φορά προς τα δεξιά και με φορά προς τα αριστερά.»

Συζήτηση:
Για να είναι το αλφαριθμητικό παλίνδρομο, θα πρέπει να είναι ίδιοι ο πρώτος και ο τελευταίος χαρακτήρας, ο δεύτερος και ο προτελευταίος κ.ο.κ. Εφόσον υπάρχουν 10 συνολικά χαρακτήρες, το τελευταίο ζεύγος που θα πρέπει να ελέγξουμε θα είναι ο $5^{ος}$ με τον $6^{ο}$ χαρακτήρα, θα γίνουν δηλαδή συνολικά 10/2=5 έλεγχοι. Εάν το μέγεθος ήταν περιττός αριθμός, τότε ο μεσαίος χαρακτήρας δεν ελέγχεται με κάποιον άλλον.
Στην περίπτωση που το μήκος του αλφαριθμητικού δεν ήταν γνωστό (πάντως ήταν μικρότερο ή ίσο προς το μέγεθος του πίνακα), θα έπρεπε πρώτα να υπολογίσουμε το

μήκος του. Εάν το μήκος του ήταν Κ, τότε θα γίνουν συνολικά Κ div 2 έλεγχοι ισότητας ζευγών χαρακτήρων.
Κατά τα λοιπά, το πρόβλημα ανήκει στην κατηγορία της απόφασης και λύνεται όπως ακριβώς το προηγούμενο.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΘΕΤΙΚΟΙ_ΑΡΙΘΜΟΙ
ΜΕΤΑΒΛΗΤΕΣ
      Χ[10]: ΧΑΡΑΚΤΗΡΕΣ
      παλίνδρομο: ΛΟΓΙΚΕΣ
      i:ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
     ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
            ΔΙΑΒΑΣΕ Χ[i]
     ΤΕΛΟΣ
     παλίνδρομο← true //έστω ότι είναι καρκινικό
     i←1
     ΟΣΟ ((i<=10 div 2) and (παλίνδρομο = true)) ΑΡΧΗ
            ΑΝ Χ[i]<> Χ[10-i+1]ΤΟΤΕ
                  παλίνδρομο←false
            i←i+1
     ΤΕΛΟΣ
     ΑΝ παλίνδρομο = true  ΤΟΤΕ
          ΓΡΑΨΕ 'το αλφαριμητικό είναι παλίνδρομο'
     ΑΛΛΙΩΣ
          ΓΡΑΨΕ 'το αλφαριθμητικό δεν είναι παλίνδρομο'
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int i, p;
   char x[11];
   for(i=1;i<=10;i++){
      scanf("%c", &x[i]);fflush(stdin);
      }
   p=1;
   i=1;
   while((i<=10/2)&&(p)){
      if(x[i]!=x[10-i+1])
         p=0;
      i++;
   }
   if(p)
     printf("palindrome");
   else
     printf("not palindrome");
   getchar();
}
```

9.2.2.5 Αντίστροφος μονοδιάστατος πίνακας

Πρόβλημα:

«Δίνεται μονοδιάστατος πίνακας 15 ακεραίων. Να αντιστραφεί ο πίνακας (το 1^{ο} στοιχείο να γίνει τελευταίο, το 2^{ο} προτελευταίο κ.ο.κ)»

Συζήτηση:

Το πρόβλημα είναι παρόμοιο με το προηγούμενο στη λογική του. Θα πρέπει να ανταλλαγούν 15 div 2 = 7 στοιχεία. Η ανταλλαγή θα γίνει με τη βοήθεια ενδιάμεσης μεταβλητής, όπως σε αντίστοιχη εφαρμογή του προηγούμενου κεφαλαίου.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΑΝΤΙΣΤΡΟΦΗ
ΜΕΤΑΒΛΗΤΕΣ
     Χ[15], I, Temp: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
     ΓΙΑ i ← 1 ΜΕΧΡΙ 15 ΑΡΧΗ
            ΔΙΑΒΑΣΕ Χ[i]
     ΤΕΛΟΣ
     ΓΙΑ i ← 1 ΜΕΧΡΙ 15 div 2 ΑΡΧΗ
            Temp← Χ[i]
            Χ[i]←Χ[15-i+1]
            Χ[15-i+1]←Temp
     ΤΕΛΟΣ
     ΓΙΑ i ← 1 ΜΕΧΡΙ 15 ΑΡΧΗ
            ΓΡΑΨΕ Χ[i]
     ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int i, temp;
   int x[16];
   for(i=1;i<=15;i++){
      scanf("%d", &x[i]);fflush(stdin);
      }
   for(i=1;i<=15/2;i++){
      temp=x[i];
      x[i]=x[15-i+1];
      x[15-i+1]=temp;
   }
   for(i=1;i<=15;i++){
      printf("%d ", x[i]);
      }
   getchar();
}
```

9.2.2.6 Πρόσθεση μονοδιάστατων πινάκων

Πρόβλημα:
«Δίνονται δύο μονοδιάστατοι πίνακες 10 ακεραίων. Να βρεθεί το άθροισμά τους»
Συζήτηση:
Το άθροισμα δύο πινάκων Α και Β μεγέθους Κ είναι ένας νέος πίνακας, έστω Γ μεγέθους επίσης Κ. Για τον υπολογισμό κάθε στοιχείου του πίνακα Γ ισχύει η σχέση: Γ[i] = A[i]+B[i], για i=1, 2, …, K.
Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΑΘΡΟΙΣΜΑ
ΜΕΤΑΒΛΗΤΕΣ
     A[10], B[10], Γ[10], i: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
          ΔΙΑΒΑΣΕ A[i], B[i]
    ΤΕΛΟΣ
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
          Γ[i]←A[i]+B[i]
    ΤΕΛΟΣ
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
          ΓΡΑΨΕ Γ[i]
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int i, a[11], b[11], c[11];
   for(i=1;i<=10;i++){
      scanf("%d", &a[i]);fflush(stdin);
      scanf("%d", &b[i]);fflush(stdin);
      }
   for(i=1;i<=10;i++){
      c[i]=a[i]+b[i];
   }
   for(i=1;i<=10;i++){
      printf("%d ", c[i]);
      }
   getchar();
}
```

9.2.2.7 Πολλαπλασιασμός δύο μονοδιάστατων πινάκων (scalar product)

Πρόβλημα:
«Δίνονται δύο μονοδιάστατοι πίνακες 10 ακεραίων. Να βρεθεί το γινόμενό τους.»
Συζήτηση
Το γινόμενο δύο πινάκων μεγέθους Κ δεν είναι ένας τρίτος πίνακας αλλά ένας αριθμός που προκύπτει ως άθροισμα γινομένων. Πιο συγκεκριμένα, πολλαπλασιάζονται όλα τα στοιχεία με τον ίδιο δείκτη και στη συνέχεια αθροίζονται. Έτσι, το τελικό αποτέλεσμα *S* του γινομένου *A* επί *B* (μεγέθους Κ), προκύπτει ως

$$S=A[1]*B[1]+A[2]*B[2]+...+A[K]*B[K] = \sum_{i=1}^{K} A[i]*B[i].$$

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΓΙΝΟΜΕΝΟ
ΜΕΤΑΒΛΗΤΕΣ
     A[10], B[10], I, S: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
          ΔΙΑΒΑΣΕ A[i], B[i]
    ΤΕΛΟΣ
    S←0
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
          S←S+A[i]*B[i]
    ΤΕΛΟΣ
    ΓΡΑΨΕ S
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int i, a[11], b[11], S=0;
   for(i=1;i<=10;i++){
      scanf("%d", &a[i]);fflush(stdin);
      scanf("%d", &b[i]);fflush(stdin);
      }
   for(i=1;i<=10;i++){
      S=S+a[i]*b[i];
   }
   printf("%d ", S);
   getchar();
}
```

9.2.2.8 Μετατροπές αριθμών

Πρόβλημα:

«Να γίνει αλγόριθμος που θα μετατρέπει έναν θετικό δεκαδικό αριθμό σε δυαδικό»

Συζήτηση:

Όπως γνωρίζουμε, η μετατροπή ενός αριθμού από το δεκαδικό σύστημα αρίθμησης στο δυαδικό σύστημα, περιλαμβάνει μια σειρά από διαιρέσεις δια της βάσης-προορισμού (2). Οι διαιρέσεις γίνονται διαδοχικά έως ότου το πηλίκο να γίνει 0. Ωστόσο, σε κάθε διαίρεση, το υπολογιζόμενο υπόλοιπο θα πρέπει να αποθηκεύεται, καθώς η τελική μορφή του αριθμού προκύπτει από την αναγραφή όλων των υπολοίπων με αντίστροφη φορά. Κατά συνέπεια, η χρήση πίνακα για τα υπόλοιπα είναι απολύτως απαραίτητη.

Το μέγεθος του πίνακα για τα υπόλοιπα εξαρτάται από το μέγεθος του αρχικού δεκαδικού αριθμού. Για παράδειγμα, αν ο δεκαδικός αριθμός είναι τριψήφιος (<=999), τότε απαιτούνται 10 δυαδικά ψηφία, ενώ αν είναι τετραψήφιος τότε απαιτούνται 14 δυαδικά ψηφία. Υπενθυμίζεται ότι ο μέγιστος δεκαδικός που μπορεί να αναπαρασταθεί με 3 δυαδικά ψηφία είναι ο 7, με 4 δυαδικά ψηφία είναι ο 15 ενώ με *κ* δυαδικά ψηφία είναι ο $2^{\kappa}-1$. Στο πρόβλημά μας, θα θεωρήσουμε ότι οι δεκαδικοί αριθμοί είναι το πολύ τριψήφιοι.

Τέλος, σημειώνεται ότι θα πρέπει να αποθηκεύουμε και το πλήθος των δυαδικών ψηφίων, το οποίο μπορεί να είναι μικρότερο ή ίσο του 10.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΜΕΤΑΤΡΟΠΗ_1
ΜΕΤΑΒΛΗΤΕΣ
    ΥΠ[10], χ, i, count: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
        ΥΠ[i] ← 0
    ΤΕΛΟΣ
    ΕΠΑΝΑΛΑΒΕ
        ΔΙΑΒΑΣΕ χ
    ΜΕΧΡΙ Χ>=0 or Χ<=999
    count←0
    ΟΣΟ χ>0 ΑΡΧΗ
        count←count+1
        ΥΠ[count]←χ mod 2
        χ←χ div 2
    ΤΕΛΟΣ
    ΓΙΑ i ← count ΜΕΧΡΙ 1 ΒΗΜΑ -1 ΑΡΧΗ
        ΓΡΑΨΕ ΥΠ[i]
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int i, a[11], count=0, x;
   for(i=1;i<=10;i++){
      a[i]=0;
   }
   do{
      scanf("%d", &x);fflush(stdin);
   }while((x<0)||(x>999));
   while(x>0){
      count++;
      a[count]=x%2;
      x=x/2;
    }
    for(i=count;i>=1;i--)
       printf("%d",a[i]);
   getchar();
}
```

Πρόβλημα:

«Να μετατραπεί ένας δυαδικός αριθμός σε δεκαδικό. Ο δυαδικός αριθμός εισάγεται ως ακέραιος»

Συζήτηση:

Στο πρόβλημα αυτό, αντί για συνεχόμενες διαιρέσεις, το τελικό αποτέλεσμα θα προκύψει ως άθροισμα γινομένων (δυαδικό ψηφίο επί σημαντική θέση). Κατά συνέπεια, είναι φανερό ότι θα πρέπει να «ξεχωρίσουμε» κάθε ψηφίο του αρχικού

αριθμού και να βρούμε τη σημαντική του θέση. Η εργασία αυτή μπορεί να γίνει με επαναλαμβανόμενες διαιρέσεις του αρχικού αριθμού δια του 10, σύμφωνα με την αντίστοιχη εφαρμογή που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Σε κάθε διαίρεση, αποθηκεύεται το υπόλοιπο σε έναν πίνακα, ενώ υπολογίζεται και το πλήθος των ξεχωριστών δυαδικών ψηφίων. Θα πρέπει λοιπόν και πάλι να υπάρχει ένα άνω όριο στο μέγεθος του αρχικού αριθμού, έτσι ώστε να δηλωθεί ο κατάλληλος πίνακας. Στο παράδειγμά μας θα χρησιμοποιήσουμε και πάλι έναν δυαδικό αριθμό 10 ψηφίων.
Σημαντικό είναι να σημειωθεί ότι με αυτόν τον τρόπο κάθε ψηφίο του δυαδικού αριθμού θα αποθηκευτεί σε έναν πίνακα, με το λιγότερο σημαντικό ψηφίο στη θέση 1, το αμέσως περισσότερο σημαντικό ψηφίο στη θέση 2 κ.ο.κ.
Για παράδειγμα, ο αριθμός *11011101*, θα αποθηκευτεί στον πίνακα ως:

1	0	1	1	1	0	1	1	0	0

Ενώ το υπολογισμένο πλήθος ψηφίων θα είναι 8.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΜΕΤΑΤΡΟΠΗ_2_10
ΜΕΤΑΒΛΗΤΕΣ
    ΥΠ[10], χ, i, count, number: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
        ΥΠ[i] ← 0
    ΤΕΛΟΣ
    ΔΙΑΒΑΣΕ x
    count←0
    ΟΣΟ χ>0 ΑΡΧΗ
        count←count+1
        ΥΠ[count]←χ mod 10
        χ←χ div 10
    ΤΕΛΟΣ
    number←0
    ΓΙΑ i ← 1 ΜΕΧΡΙ count ΑΡΧΗ
        number←number+ΥΠ[i]*2^(i-1)
    ΤΕΛΟΣ
    ΓΡΑΨΕ number
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int i, a[11], count=0, x, number;
   for(i=1;i<=10;i++){
      a[i]=0;
   }
   scanf("%d", &x);fflush(stdin);
   while(x>0){
      count++;
      a[count]=x%10;
      x=x/10;
    }
    number=0;
    for(i=1;i<=count;i++)
        number←number+a[i]*pow(2, i-1);
    printf("%d", number);
   getchar();
}
```

Οι δύο αλγόριθμοι που προαναφέρθηκαν μπορούν να χρησιμοποιηθούν, με κατάλληλες τροποποιήσεις, για όλες τις μετατροπές ανάμεσα σε αριθμητικά συστήματα με βάση b<=10. Ωστόσο, εάν ένα από τα δύο συστήματα βρίσκεται σε μεγαλύτερη βάση, χρειάζεται επιπλέον ένας τρόπος αποθήκευσης των επιπλέον ψηφίων. Για παράδειγμα, σε μια μετατροπή από το δεκαδικό στο δεκαεξαδικό σύστημα, θα πρέπει τα δεκαεξαδικά ψηφία να βρίσκονται αποθηκευμένα σε κάποιον άλλον βοηθητικό πίνακα με τη μορφή χαρακτήρων. Το αποτέλεσμα θα πρέπει επίσης να αποθηκεύεται σε πίνακα χαρακτήρων, όπως φαίνεται στον ακόλουθο αλγόριθμο:

```
ΑΛΓΟΡΙΘΜΟΣ ΜΕΤΑΤΡΟΠΗ_10_16
ΜΕΤΑΒΛΗΤΕΣ
    χ, i, count: ΑΚΕΡΑΙΕΣ
    ΥΠ[10], HEX[16]:ΧΑΡΑΚΤΗΡΕΣ
ΑΡΧΗ
    HEX[1]←'0'
    HEX[2]←'1'
    ……
    HEX[10]←'A'
    HEX[11]←'B'
```

```
#include <stdio.h>
int main(void){
   int i, count=0, x;
  char yp[11],hex[17];
  hex[1]='0';hex[2]='1';hex[3]='2'; hex[4]='3'; hex[5]='4';
  hex[6]='5';hex[7]='6'; hex[8]='7'; hex[9]='8'; hex[10]='9';
  hex[11]='A'; hex[12]='B'; hex[13]='C'; hex[14]='D';
  hex[15]='E'; hex[16]='F';
   for(i=1;i<=10;i++){
      yp[i]='0';
```

```
    ….
    HEX[15]←'F'
    ΓΙΑ i ← 1 ΜΕΧΡΙ 10 ΑΡΧΗ
          ΥΠ[i] ← '0'
    ΤΕΛΟΣ
    ΔΙΑΒΑΣΕ x

    count←0
    ΟΣΟ χ>0 ΑΡΧΗ
        count←count+1
        ΥΠ[count]←HEX[χ mod 16+1]
        χ←χ div 16
    ΤΕΛΟΣ
    ΓΙΑ i ← count ΜΕΧΡΙ 1 ΒΗΜΑ -1 ΑΡΧΗ
          ΓΡΑΨΕ ΥΠ[i]
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
    }
    scanf("%d", &x);fflush(stdin);
    while(x>0){
        count++;
        yp[count]=hex[x%16+1];
        x=x/16;
    }
    for(i=count;i>=1;i--)
        printf("%c", yp[i]);
    getchar();
}
```

9.2.2.9 Βασικοί υπολογισμοί σε δισδιάστατους πίνακες

Πρόβλημα:

«Να εισαχθεί δισδιάστατος πίνακας ακεραίων μεγέθους 4X5 (4 γραμμών και 5 στηλών). Να υπολογιστεί το άθροισμα όλων των στοιχείων, το άθροισμα κάθε γραμμής και ο μέσος όρος κάθε στήλης. Να υπολογιστεί ο μέγιστος κάθε άρτιας γραμμής και ο ελάχιστος κάθε περιττής στήλης».

Συζήτηση:

Το άθροισμα όλων των στοιχείων του πίνακα δεν εμφανίζει ιδιαίτερη δυσκολία και θα υπολογιστεί διατρέχοντας όλα τα στοιχεία του. Για κάθε γραμμή, υπάρχει ένα άθροισμα, κατά συνέπεια θα υπολογιστούν 4 αθροίσματα για τις γραμμές. Όμοια, για κάθε στήλη υπάρχουν 5 αθροίσματα και 5 μέσοι όροι. Κάθε αποτέλεσμα μπορεί να εμφανίζεται μόλις υπολογιστεί. Τα μέγιστα και ελάχιστα θα υπολογιστούν με παρόμοιο τρόπο όπως και στους μονοδιάστατους πίνακες.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΥΠΟΛΟΓΙΣΜΟΙ
ΜΕΤΑΒΛΗΤΕΣ
    A[4,5], i, j, S, max, min: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ 4
        ΓΙΑ j ← 1 ΜΕΧΡΙ 5
            ΔΙΑΒΑΣΕ(A[i, j])
    S←0  //αρχικοποίηση συνολικού αθροίσματος
    ΓΙΑ i ← 1 ΜΕΧΡΙ 4
        ΓΙΑ j ← 1 ΜΕΧΡΙ 5
            S←S+A[i, j]
    ΓΡΑΨΕ S  //το άθροισμα όλων των στοιχείων

    ΓΙΑ i ← 1 ΜΕΧΡΙ 4 ΑΡΧΗ //για κάθε γραμμή…
        S←0 //μηδενισμός αθροίσματος i-στης γραμμής
        ΓΙΑ j ← 1 ΜΕΧΡΙ 5 ΑΡΧΗ
            S←S+A[i, j]
        ΤΕΛΟΣ
        ΓΡΑΨΕ S //το άθροισμα της i γραμμής
    ΤΕΛΟΣ

    ΓΙΑ j ← 1 ΜΕΧΡΙ 5 ΑΡΧΗ  //για κάθε στήλη
        S←0 //μηδενισμός αθροίσματος i-στης γραμμής
        ΓΙΑ i ← 1 ΜΕΧΡΙ 4 ΑΡΧΗ
            S←S+A[i, j]
```

```
#include <stdio.h>
int main(void){
    int i, j, a[5][6], S, max, min;
    for(i=1;i<=4;i++)
        for(j=1;j<=5;j++){
            scanf("%d", &a[i][j]);fflush(stdin)
    }
    S=0;
    for(i=1;i<=4;i++)
        for(j=1;j<=5;j++)
            S+=a[i][j];
    printf("%d\n", S);

    for(i=1;i<=4;i++){
        S=0;
        for(j=1;j<=5;j++)
            S+=a[i][j];
        printf("%d\n", S);
    }

    for(j=1;j<=5;j++){
        S=0;
        for(i=1;i<=5ij++)
            S+=a[i][j];
        printf("%4.2f\n", S/4);
```

```
        ΤΕΛΟΣ
        ΓΡΑΨΕ S/4 //ο μέσος όρος της j-στής στήλης
    ΤΕΛΟΣ

    ΓΙΑ i ← 1 ΜΕΧΡΙ 4 ΑΡΧΗ //για κάθε γραμμή…
        ΑΝ i mod 2=0 ΤΟΤΕ ΑΡΧΗ// άρτια γραμμή
            max←A[i, 1]
            ΓΙΑ j ← 2 ΜΕΧΡΙ 5
                ΑΝ A[i, j]>max ΤΟΤΕ
                    max←A[i, j]
            ΓΡΑΨΕ max
        ΤΕΛΟΣ  //του εξωτερικού ΑΝ
    ΤΕΛΟΣ

    ΓΙΑ j ← 1 ΜΕΧΡΙ 5 ΑΡΧΗ //για κάθε στήλη…
        ΑΝ j mod 2=1 ΤΟΤΕ ΑΡΧΗ// περιττή στήλη
            min←A[1, j]
            ΓΙΑ i ← 2 ΜΕΧΡΙ 4
                ΑΝ A[i, j]<min ΤΟΤΕ
                    min←A[i, j]
            ΓΡΑΨΕ min
        ΤΕΛΟΣ  //του εξωτερικού ΑΝ
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
    }

    for(i=1;i<=4;i++){
        if(i%2==0){
            max=a[i][1];
            for(j=2;j<=5;j++)
                if(a[i][j]>max)
                    max=a[i][j];
            printf("%d\n", max);
        }
    }

    for(j=1;j<=5;j++){
        if(j%2==1){
            min=a[1][j];
            for(j=2;j<=5;j++)
                if(a[i][j]<min)
                    min=a[i][j];
            printf("%d\n", min);
        }
    }

    getchar();
}
```

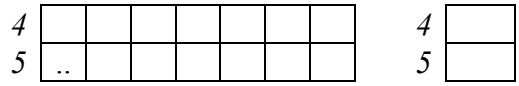
Παρατηρήσεις:
Ιδιαίτερη προσοχή θα πρέπει να δοθεί στην κατάλληλη αρχικοποίηση των τιμών που υπολογίζονται. Έτσι, το άθροισμα όλων των στοιχείων του πίνακα θα αρχικοποιηθεί σε 0 πριν από κάθε υπολογισμό, ενώ το άθροισμα κάθε γραμμής θα πρέπει να αρχικοποιηθεί σε 0 μόλις αρχίζει η επεξεργασία αυτής της γραμμής, δηλαδή μόλις πριν ακριβώς αρχίσει η εσωτερική επανάληψη.

9.2.2.10 Χρήση βοηθητικών πινάκων

Πρόβλημα:
«Δίνεται δισδιάστατος πίνακας ακεραίων 5Χ7. Να βρεθούν οι μέσοι όροι κάθε γραμμής. Να υπολογιστεί ο μεγαλύτερος μέσος όρος και σε ποια γραμμή συναντάται.»
Συζήτηση:
Το πρόβλημα μοιάζει με το προηγούμενο στους υπολογισμούς. Η εύρεση των μέσων όρων είναι πλέον απλή, ενώ ταυτόχρονα μπορεί να υπολογίζεται και ο μέγιστος. Ωστόσο, το τελευταίο ερώτημα αλλάζει τις προϋποθέσεις. Για να βρεθεί η γραμμή (ή οι γραμμές) όπου συναντάται ο μέγιστος μέσος όρος, θα πρέπει όλοι οι υπολογισμένοι μέσοι όροι να αποθηκευτούν σε κάποια δομή δεδομένων, δηλαδή σε έναν νέο πίνακα. Εφόσον υπάρχει ένας μέσος όρος σε κάθε γραμμή, απαιτείται ένας μονοδιάστατος πίνακας, 5 στοιχείων, στον οποίο θα αποθηκεύται ο μέσος όρος κάθε γραμμής. Σε αυτόν τον πίνακα θα υπολογιστεί η μέγιστη τιμή και θα αναζητηθεί η γραμμή. Σχηματικά, οι πίνακες φαίνονται στη συνέχεια:

	1	*2*	*3*	*4*	*5*	*6*	*7*
1	2	3	5	2	3	8	5
2	..	..					
3	..						

1	7,0
2	
3	



Πίνακας Α με δεδομένα Πίνακας ΜΟ με τους μέσους όρους κάθε γραμμής

Οι δύο πίνακες θα λειτουργούν «παράλληλα» με την έννοια ότι υπάρχει μία λογική (και όχι φυσική) σύνδεση μεταξύ τους. Η χρήση τέτοιων παράλληλων πινάκων είναι πολύ συχνή, όχι μόνο σε περιπτώσεις όπως αυτή που περιγράφεται πιο πάνω αλλά και αλλού, όπως για παράδειγμα στην περίπτωση που πρέπει να αποθηκεύουμε δεδομένα διαφορετικού τύπου για μία οντότητα. Έτσι, εάν απαιτείται να αποθηκευτούν Κ ονόματα μαζί με τις αντίστοιχες Κ ηλικίες τους, η χρήση δύο μονοδιάστατων πινάκων μεγέθους Κ είναι η μοναδική λύση (προς το παρόν τουλάχιστον). Σε κάθε περίπτωση, η σύνδεση μεταξύ των πινάκων δημιουργείται από τον προγραμματιστή.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ Β_Π
ΜΕΤΑΒΛΗΤΕΣ
     A[5,7], i, j: ΑΚΕΡΑΙΕΣ
     MO[5],max: ΠΡΑΓΜΑΤΙΚΕΣ
ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ 5
          ΓΙΑ j ← 1 ΜΕΧΡΙ 7
               ΔΙΑΒΑΣΕ(A[i, j])
    ΓΙΑ i ← 1 ΜΕΧΡΙ 5 ΑΡΧΗ //για κάθε γραμμή…
          MO[i]←0
          ΓΙΑ j ← 1 ΜΕΧΡΙ 7
               MO[i]←MO[i]+A[i, j]
          MO[i]←MO[i]/7
    ΤΕΛΟΣ
    Max←MO[1]
    ΓΙΑ i ← 2 ΜΕΧΡΙ 5
          ΑΝ MO[i]>max ΤΟΤΕ
               max←MO[i]

     ΓΙΑ i ← 1 ΜΕΧΡΙ 5 ΑΡΧΗ
          ΑΝ MO[i]=max ΤΟΤΕ
               ΓΡΑΨΕ 'Ο ',max,' βρέθηκε στη γραμμή:',i
     ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
int main(void){
   int i, j, a[6][8];
   float MO[6],max;
   for(i=1;i<=5;i++)
      for(i=1;i<=7;i++){
         scanf("%d", &a[i][j]);fflush(stdin)
   }
   for(i=1;i<=5;i++){
      MO[i]=0;
      for(j=1;j<=7;j++)
               MO[i]+=a[i][j];
      MO[i]=MO[i]/7
   }
   max=MO[1];
   for(i=2;i<=5;i++)
      if(MO[i]>max)
         max=MO[i];

   for(i=1;i<=5;i++)
      if(max==MO[i])
         printf("found %d in line %d\n", max, i);

  getchar();
}
```

9.2.2.11 Ένα «πραγματικό» παράδειγμα

Πρόβλημα:

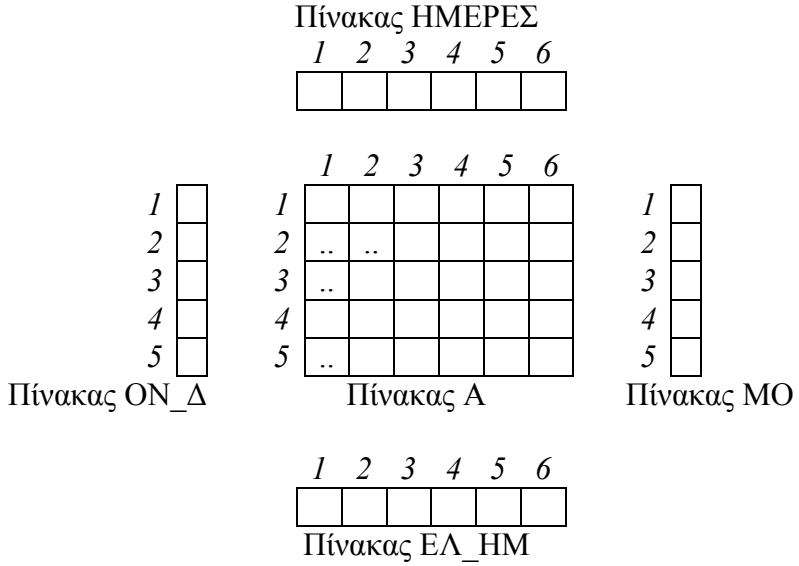
«Σε ένα πολυκατάστημα υπάρχουν 5 διαφορετικά μικρότερα υποκαταστήματα (shop-in-shop). Κάθε ένα από αυτά έχει έναν υπάλληλο-διευθυντή. Η κεντρική διοίκηση του υποκαταστήματος, προκειμένου να επιβραβεύσει την εργασία των μικρών καταστημάτων, προβαίνει στην ακόλουθη στρατηγική: Αποθηκεύει τις πωλήσεις (τον τζίρο) κάθε υποκαταστήματος για κάθε ημέρα της εργάσιμης εβδομάδας (Δευτέρα έως Σάββατο). Στη συνέχεια, υπολογίζει το μέσο όρο πωλήσεων κάθε υποκαταστήματος. Σε κάθε διευθυντή υποκαταστήματος που ο μέσος όρος πωλήσεών του υπερβαίνει ένα δεδομένο ποσό-στόχο, δίνεται επιπλέον παροχή ενός γνωστού ποσού (έστω 300€). Εάν επιπλέον κάποιο υποκατάστημα πετυχαίνει το μέγιστο μέσο όρο, τότε η παροχή αυτή διπλασιάζεται. Η κεντρική διοίκηση, προκειμένου να προβεί σε καλύτερη διαχείριση του

προσωπικού, υπολογίζει την ελάχιστη πώληση για κάθε ημέρα της εβδομάδας. Στα υποκαταστήματα που έχουν πωλήσεις ίσες με τη ελάχιστη πώληση της ημέρας, φροντίζει στην επόμενη ημέρα να μετακινεί προσωπικό από άλλα τμήματα της επιχείρησης. Τέλος η κεντρική διοίκηση υπολογίζει τη μέρα ή τις ημέρες που είχαν τις μικρότερες απόλυτες πωλήσεις από όλα τα υποκαταστήματα σε όλες τις ημέρες. Εάν αυτές αντιστοιχούν σε υποκατάστημα του οποίου ο μέσος εβδομαδιαίος όρος πωλήσεων ξεπερνά τον εβδομαδιαίο στόχο, τότε δεν υπάρχει καμία επίπτωση. Εάν όμως δεν τον ξεπερνούν, τότε ο διευθυντής του υποκαταστήματος υφίσταται διοικητικές κυρώσεις (προειδοποιείται ή απολύεται…).

Ζητείται να δημιουργηθεί ένα ολοκληρωμένο πρόγραμμα που έχοντας διαβάσει τα απαραίτητα δεδομένα, θα βοηθά την κεντρική διοίκηση να λάβει τις απαραίτητες αποφάσεις, υπολογίζοντας και εκτυπώνοντας τις απαραίτητες ποσότητες και μηνύματα.»

Συζήτηση:

Σε αυτό το πρόβλημα είναι προφανές ότι θα χρειαστεί να χρησιμοποιηθούν αρκετοί παράλληλοι πίνακες. Έτσι, χρειάζεται ένας μονοδιάστατος πίνακας, έστω ΟΝ_Δ, με τα πέντε ονόματα των διευθυντών. Χρειάζεται επίσης ένας δισδιάστατος πίνακας, έστω Π, 5Χ6, στον οποίο θα αποθηκεύεται ο τζίρος κάθε υποκαταστήματος για κάθε ημέρα της εβδομάδας. Ένας μονοδιάστατος πίνακας ΜΟ με 5 θέσεις θα χρησιμοποιηθεί για τους μέσους όρους πωλήσεων κάθε υποκαταστήματος, ενώ ένας ακόμη μονοδιάστατος πίνακας ΕΛ_ΗΜ, 6 θέσεων, θα έχει την ελάχιστη τιμή πωλήσεων κάθε ημέρας. Βοηθητικά, θα χρησιμοποιηθεί και ένας πίνακας ΗΜΕΡΕΣ, 6 θέσεων, που θα περιλαμβάνει τα ονόματα των ημερών της εβδομάδας.. Σχηματικά, οι πίνακες φαίνονται παρακάτω:



Τα δεδομένα που θα εισαχθούν αφορούν στους πίνακες ΟΝ_Δ και Α. Ο πίνακας ΗΜΕΡΕΣ είναι καλύτερο να αρχικοποιηθεί εντός του προγράμματος, ενώ θα υπολογιστούν οι τιμές των στοιχείων των ΜΟ και ΜΟ_ΗΜ. Σε ένα τέτοιο

πρόγραμμα, σκόπιμο είναι να χρησιμοποιηθεί αμυντικός προγραμματισμός στις τιμές του πίνακα Α, οι οποίες θα πρέπει βέβαια να είναι μη αρνητικές.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΠΩΛΗΣΕΙΣ
ΜΕΤΑΒΛΗΤΕΣ
     Α[5,6], ΕΛ_ΗΜ[6], i, j,Η,min,ποσο: ΑΚΕΡΑΙΕΣ
     ΜΟ[5], max,ΣΤΟΧΟΣ: ΠΡΑΓΜΑΤΙΚΕΣ
     ΟΝ[5], ΗΜΕΡΕΣ[6]: ΧΑΡΑΚΤΗΡΕΣ
ΑΡΧΗ
  ΗΜΕΡΕΣ[1]←'ΔΕΥΤΕΡΑ'; ΗΜΕΡΕΣ[1]←'ΤΡΙΤΗ';
  ΗΜΕΡΕΣ[1]←'ΤΕΤΑΡΤΗ'; ΗΜΕΡΕΣ[1]←'ΠΕΜΠΤΗ';
  ΗΜΕΡΕΣ[1]←'ΠΑΡΑΣΚΕΥΗ'; ΗΜΕΡΕΣ[1]←'ΣΑΒΒΑΤΟ';
    ΓΙΑ i ← 1 ΜΕΧΡΙ 5
          ΓΙΑ j ← 1 ΜΕΧΡΙ 6
             ΕΠΑΝΑΛΑΒΕ
               ΔΙΑΒΑΣΕ(Α[i, j])
            ΜΕΧΡΙ Α[i, j]>=0)
    ΓΙΑ i ← 1 ΜΕΧΡΙ 5
       ΔΙΑΒΑΣΕ ΟΝ[i]

    ΓΙΑ i ← 1 ΜΕΧΡΙ 5 ΑΡΧΗ //για κάθε γραμμή…
          ΜΟ[i]←0
          ΓΙΑ j ← 1 ΜΕΧΡΙ 6
               ΜΟ[i]←ΜΟ[i]+Α[i, j]
          ΜΟ[i]←ΜΟ[i]/6
    ΤΕΛΟΣ

     ΔΙΑΒΑΣΕ ΣΤΟΧΟΣ
     max←ΜΟ[1]
    ΓΙΑ i ← 2 ΜΕΧΡΙ 5
          ΑΝ ΜΟ[i]>max ΤΟΤΕ
               max←ΜΟ[i]

    ΓΙΑ i ← 1 ΜΕΧΡΙ 5 ΑΡΧΗ
         ποσο←0
         ΑΝ ΜΟ[i]>ΣΤΟΧΟΣ ΤΟΤΕ
            ποσο←300
         ΑΝ ΜΟ[i]=max ΤΟΤΕ
              ποσό←ποσο*2
          ΑΝ ποσο>0 ΤΟΤΕ
               ΓΡΑΨΕ 'Χορήγηση ',ποσο,'€ στον',ΟΝ[ι]
     ΤΕΛΟΣ

     ΓΙΑ j ← 1 ΜΕΧΡΙ 6 ΑΡΧΗ //για κάθε στήλη…
          ΕΛ_ΗΜ[j]←Α[1,j]
          ΓΙΑ i ← 1 ΜΕΧΡΙ 5
               ΑΝ Α[i, j]<ΕΛ_ΗΜ[j] ΤΟΤΕ
                    ΕΛ_ΗΜ[j]←Α[i, j]
    ΤΕΛΟΣ
    ΓΙΑ j ← 1 ΜΕΧΡΙ 6 //για κάθε στήλη
        ΓΙΑ i ← 1 ΜΕΧΡΙ 5
          ΑΝ Α[i, j]=ΕΛ_ΗΜ[j] ΤΟΤΕ ΑΡΧΗ
             ΑΝ j=6 ΤΟΤΕ
                Η←1
             ΑΛΛΙΩΣ
                Η←j
             ΓΡΑΨΕ 'Ο ',ΟΝ[i],' χρειάζεται προσωπικό'
             ΓΡΑΨΕ ' για την ', ΗΜΕΡΕΣ[Η]
          ΤΕΛΟΣ
   Min←Α[1,1]
   ΓΙΑ i ← 1 ΜΕΧΡΙ 5
         ΓΙΑ j ← 1 ΜΕΧΡΙ 6
               ΑΝ Α[i, j]<min ΤΟΤΕ
                  min←Α[i, j]
   ΓΙΑ i ← 1 ΜΕΧΡΙ 5
```

```
#include <stdio.h>
#include <string.h>
int main(void){
   int i, j, h, min, EL_HM[7], a[6][7],poso;
   float MO[6], max, STOXOS;
   char ON[6][30],HMERES[7][10],o[30];

strcpy(HMERES[1],"DEYTERA");                    strc-
py(HMERES[2],"TRITH");
strc-
py(HMERES[3],"TETARTH");strcpy(HMERES[4],"PEM
PTH");strcpy(HMERES[5],"PARASKEYH");            strc-
py(HMERES[6],"SABBATO");
  for(i=1;i<=5;i++)
     for(j=1;j<=6;j++)
       do{
        scanf("%d", &a[i][j]);fflush(stdin);
       }while(a[i][j]<0);
  for(i=1;i<=5;i++){
     scanf("%s" o);fflush(stdin);
     strcpy(ON[i],o);
  }

  for(i=1;i<=5;i++){
     MO[i]=0;
     for(j=1;j<=6;j++)
              MO[i]+=a[i][j];
     MO[i]=MO[i]/6;
  }
  scanf("%d", &STOXOS);
  max=MO[1];
  for(i=2;i<=5;i++)
     if(MO[i]>max)
        max=MO[i];

  for(i=1;i<=5;i++){
     poso=0;
     if(MO[i]>STOXOS)
       poso=300;
     if(MO[i]==max)
       poso*=2;
     if(poso>0)
        printf("give  %d to %s\n", poso, ON[i]);
  }

  for(j=1;j<=6;j++){
    EL_HM[j]=a[1][j];
    for(i=1;i<=5;i++)
       if(a[i][j]<EL_HM[j])
         EL_HM[j]=a[i][j];
  }
  for(j=1;j<=6;j++)
    for(i=1;i<=5;i++)
      if(a[i][j]==EL_HM[j]) {
        if(j==6)
          h=1;
        else
          h=j;
        printf("%s needs people ",ON[i]);
        printf("for day %s\n", HMERES[h]);
  }
 min=a[1][1];
```

ΓΙΑ j ← 1 **ΜΕΧΡΙ** 6 **ΑΝ** (A[i, j]=min **and** MO[i]<ΣΤΟΧΟΣ) **ΤΟΤΕ** **ΓΡΑΨΕ** 'Προσοχή στον ',ON[i] **ΤΕΛΟΣ**	for(i=1;i<=5;i++) for(j=1;j<=6;j++) if(a[i][j]<min) min=a[i][j]; for(i=1;i<=5;i++) for(j=1;j<=6;j++) if((a[i][j]==min)&&(MO[i]<STOXOS)) printf("warning to %s",ON[i]); getchar(); }

9.2.2.12 Άθροισμα και πολλαπλασιασμός δισδιάστατων πινάκων

Πρόβλημα:

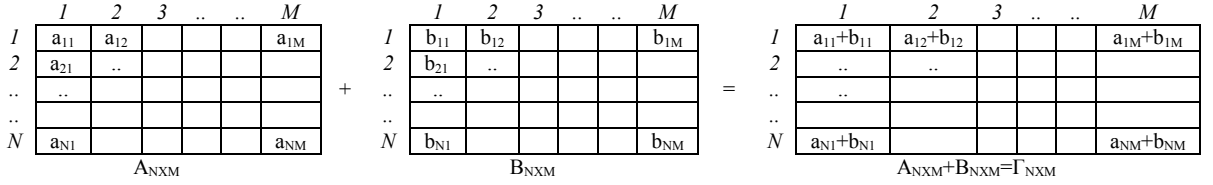
«Δίνονται δύο πίνακες ακεραίων A_{NXM} και B_{NXM}. Να βρεθεί το άθροισμά τους. Δίνεται πίνακας ακεραίων $Δ_{MXK}$. Να βρεθεί το γινόμενο των πινάκων Α Χ Δ.»

Συζήτηση:

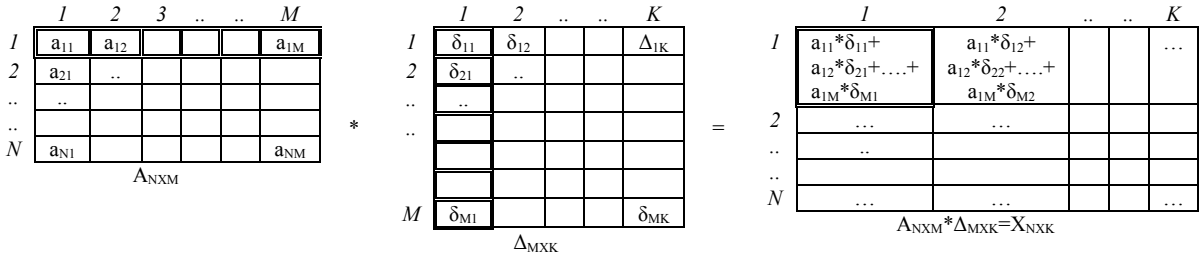
Το άθροισμα των δύο δισδιάστατων πινάκων A_{NXM} *και* B_{NXM} με ίδιο τύπο δεδομένων, είναι ένα νέος πίνακας $Γ_{NXM}$. Κάθε στοιχείο του Γ προκύπτει από το άθροισμα των αντίστοιχων στοιχείων των Α και Β. Ισχύει δηλαδή: Γ[i, j]=A[i,j]+B[i,j], για i=1,2,..N και j=1,2,…M.

Το γινόμενο δύο δισδιάστατων πινάκων προκύπτει μόνο όταν το πλήθος των στηλών του πρώτου ισούται με το πλήθος των γραμμών του δεύτερου. Έτσι, ο πολλαπλασιασμός των πινάκων A_{NXM} και $Δ_{MXK}$ είναι εφικτός, ενώ ο πολλαπλασιασμός των A_{NXM} και B_{NXM} **δεν** είναι εφικτός. Το γινόμενο των Α και Δ είναι ένας νέος πίνακας X_{NXK}. Κάθε στοιχείο X[i,j] του πίνακα Χ προκύπτει ως το γινόμενο των μονοδιάστατων πινάκων που καθορίζονται από την i-στη γραμμή του Α και τη j-στη στήλη του Β.

Σχηματικά, το άθροισμα των πινάκων A_{NXM} *και* B_{NXM} και το γινόμενο των πινάκων A_{NXM} και $Δ_{MXK}$ φαίνεται παρακάτω, όπου για λόγους συντομογραφίας, χρησιμοποιείται το a_{ij} για να αναπαραστήσει το A[i,j]:



Πρόσθεση Πινάκων



Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΠΡΟΣΘΕΣΗ_ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ
ΣΤΑΘΕΡΕΣ
  N=4, M=5, K=6
ΜΕΤΑΒΛΗΤΕΣ
     A[N,M], B[N,M], Γ[N,M], Δ[M,K], X[N,K], i, j,l: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          ΓΙΑ j ← 1 ΜΕΧΡΙ M
               ΔΙΑΒΑΣΕ(A[i, j], B[i,j])
    ΓΙΑ i ← 1 ΜΕΧΡΙ M
          ΓΙΑ j ← 1 ΜΕΧΡΙ K
               ΔΙΑΒΑΣΕ(Δ[i,j])

    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          ΓΙΑ j ← 1 ΜΕΧΡΙ M ΑΡΧΗ
               Γ[i,j]←A[i, j]+ B[i,j]  //άθροισμα πινάκων
               ΓΡΑΨΕ Γ[i,j]
          ΤΕΛΟΣ

    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          ΓΙΑ j ← 1 ΜΕΧΡΙ K ΑΡΧΗ
               X[i,j]←0
               ΓΙΑ l←1 ΜΕΧΡΙ M //πολλαπλασιασμός
                    X[i,j]← X[i,j]+A[i,l]*Δ[l,j]
           ΤΕΛΟΣ

    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          ΓΙΑ j ← 1 ΜΕΧΡΙ K
                ΓΡΑΨΕ X[i,j]
ΤΕΛΟΣ
```

```
#include <stdio.h>
#define N 4
#define M 5
#define K 6
int main(void){
   int i, j, l, a[N+1][M+1], b[N+1][M+1], c[N+1][M+1];
   int d[M+1][K+1],x[N+1][K+1];
  for(i=1;i<=N;i++)
      for(j=1;j<=M;j++){
         scanf("%d %d", &a[i][j], &b[i][j]);fflush(stdin);}
   for(i=1;i<=M;i++)
      for(j=1;j<=K;j++){
         scanf("%d", &d[i][j]);fflush(stdin);}
    for(i=1;i<=N;i++){
      for(j=1;j<=M;j++){
         c[i][j]=a[i][j]+b[i][j];printf("%d ",c[i][j]);}
      printf("\n");}

    for(i=1;i<=N;i++)
      for(j=1;j<=K;j++){
         x[i][j]=0;
         for(l=1;l<=M;l++)
            x[i][j]=x[i][j]+a[i][l]*d[l][j];
      }
      for(i=1;i<=N;i++){
         for(j=1;j<=K;j++)
           printf("%d  ",x[i][j]);
         printf("n");
      }
  getchar();
}
```

9.2.2.13 Εργασίες σε τετραγωνικούς πίνακες

Πρόβλημα:

«Δίνεται τετραγωνικός πίνακας ακεραίων Ν Χ Ν. Να βρεθεί το άθροισμα των στοιχείων της κυρίας διαγωνίου του. Να βρεθεί το γινόμενο των στοιχείων της δευτερεύουσας διαγωνίου του. Να ελεγχθεί εάν ο πίνακας είναι συμμετρικός, άνω ή κάτω τριγωνικός ή κεντροσυμμετρικός. Εάν ο πίνακας δεν είναι συμμετρικός, τότε να αντιστραφεί (τα στοιχεία της 1^{ης} γραμμής να ανήκουν στην 1^{η} στήλη, της 2^{ης} γραμμής στη 2^{η} στήλη κ.ο.κ).»

Συζήτηση:

Σε έναν τετραγωνικό πίνακα Α μεγέθους ΝΧΝ, τα στοιχεία της κύριας διαγωνίου έχουν τον ίδιο δείκτη γραμμής και στήλης, είναι δηλαδή της μορφής A[i,i], i=1,2…N. Για τα στοιχεία της δευτερεύουσας διαγωνίου η σχέση που ισχύει στους δείκτες κάθε στοιχείου προκύπτει από απλή παρατήρηση της δομής του τετραγωνικού πίνακα και είναι A[i, N-i+1]. Επιπλέον, ένας τέτοιος πίνακας είναι συμμετρικός, όταν A[i,j]=A[j,i], για κάθε i,j=1,2…N. Σε έναν άνω τριγωνικό πίνακα, ισχύει A[i,j]=0, για i>j, για κάθε i, j=1,2,…N. Τέλος σε ένα κεντροσυμμετρικό πίνακα, ισχύει A[i,

j]=A[N-i+1, N-j+1], i, j = 1, 2, …N. Η αντιστροφή του πίνακα θα γίνει στοιχείο προς στοιχείο με τη χρήση ενδιάμεσης μεταβλητής. Η αντιστροφή έχει νόημα μόνο όταν ο πίνακας δεν είναι συμμετρικός, καθώς είναι φανερό ότι ο αντίστροφος ενός συμμετρικού πίνακα είναι ο ίδιος ο πίνακας.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΤΕΤΡΑΓΩΝΙΚΟΙ_ΠΙΝΑΚΕΣ
ΣΤΑΘΕΡΕΣ
  N=5
ΜΕΤΑΒΛΗΤΕΣ
     A[N,N], i, j, temp, S, P: ΑΚΕΡΑΙΕΣ
   SYM, KENTR, A_TR, K_TR:ΛΟΓΙΚΕΣ
ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          ΓΙΑ j ← 1 ΜΕΧΡΙ N
               ΔΙΑΒΑΣΕ(A[i, j])
    S←0
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          S←S+A[i,i] //άθροισμα στοιχείων κυρίας διαγωνίου
    ΓΡΑΨΕ S
     P←1
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          P←P*A[i,N-i+1] //γινόμενο δευτερεύουσας διαγ.
    ΓΡΑΨΕ P

    SYM←true
    ΓΙΑ i ← 1 ΜΕΧΡΙ N-1
        ΓΙΑ j ← i+1 ΜΕΧΡΙ N
             ΑΝ A[i,j]<>A[j,i] ΤΟΤΕ
                SYM←false
//ελέγχονται μόνο τα στοιχεία άνω της κυρίας διαγωνίου
     ΑΝ SYM=true ΤΟΤΕ
        ΓΡΑΨΕ 'ΣΥΜΜΕΤΡΙΚΟΣ ΠΙΝΑΚΑΣ'
     ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'ΜΗ ΣΥΜΜΕΤΡΙΚΟΣ ΠΙΝΑΚΑΣ'

    A_TR←true
    ΓΙΑ i ← 2 ΜΕΧΡΙ N
        ΓΙΑ j ← 1 ΜΕΧΡΙ i-1
             ΑΝ A[i,j]<>0 ΤΟΤΕ
                A_TR←false
//ελέγχονται μόνο τα στοιχεία κάτω της κυρίας διαγωνίου
     ΑΝ A_TR=true ΤΟΤΕ
        ΓΡΑΨΕ 'ΑΝΩ ΤΡΙΓΩΝΙΚΟΣ ΠΙΝΑΚΑΣ'
     ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'ΟΧΙ ΑΝΩ ΤΡΙΓΩΝΙΚΟΣ ΠΙΝΑΚΑΣ'

    K_TR←true
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
        ΓΙΑ j ← 1 ΜΕΧΡΙ N
             ΑΝ (i<j and A[i,j]<>0) ΤΟΤΕ
                K_TR←false
//ελέγχονται μόνο τα στοιχεία άνω της κυρίας διαγωνίου
     ΑΝ K_TR=true ΤΟΤΕ
        ΓΡΑΨΕ 'ΚΑΤΩ ΤΡΙΓΩΝΙΚΟΣ ΠΙΝΑΚΑΣ'
     ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'ΟΧΙ ΚΑΤΩ ΤΡΙΓΩΝΙΚΟΣ ΠΙΝΑΚΑΣ'

    KENTR←true
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
        ΓΙΑ j ← 1 ΜΕΧΡΙ N
             ΑΝ A[i,j]<>A[N-i+1,N-j+1] ΤΟΤΕ
                KENTR←false
//ελέγχονται όλα τα στοιχεία από δύο φορές!!!!
     ΑΝ KENTR=true ΤΟΤΕ
```

```
#include <stdio.h>
#define N 4
int main(void){
   int i, j, temp,s,p,sym,kentr,a_tr,k_tr;int a[N+1][N+1];
  for(i=1;i<=N;i++)
      for(j=1;j<=N;j++){
         scanf("%d", &a[i][j]);fflush(stdin);}
  s=0;
  for(i=1;i<=N;i++)
     s+=a[i][i];
  printf("\nsum is:%d",s);
  p=1;
   for(i=1;i<=N;i++)
     p*=a[i][N-i+1];
  printf("\nproduct is:%d",p);

  sym=1;
   for(i=1;i<=N-1;i++)
     for(j=i+1;j<=N;j++)
        if(a[i][j]!=a[j][i])
          sym=0;
   if(sym)
      printf("\nsymmetric");
   else
      printf("\nnon symmetric");

   a_tr=1;
   for(i=2;i<=N;i++)
        for(j=1;j<=i-1;j++)
            if(a[i][j]!=0)
               a_tr=0;
    if(a_tr)
      printf("\nupper triangular");
   else
     printf("\nnon upper triangular");

   k_tr=1;
   for(i=1;i<=N;i++)
        for(j=1;j<=N;j++)
            if((i<j)&&(a[i][j]!=0))
               k_tr=0;
    if(k_tr)
      printf("\nlower triangular");
   else
     printf("\nnon lower triangular");

   kentr=1;
   for(i=1;i<=N;i++)
        for(j=1;j<=N;j++)
            if(a[i][j]!=a[N-i+1][N-j+1])
               kentr=0;
    if(kentr)
      printf("\ncentrosymmetric\n");
   else
     printf("\nnon centrosymmetric\n");

   if(sym)
      for(i=1;i<=N-1;i++)
        for(j=i+1;j<=N;j++){
```

```
        ΓΡΑΨΕ 'ΚΕΝΤΡΟΣΥΜΜΕΤΡΙΚΟΣ ΠΙΝΑΚΑΣ'
    ΑΛΛΙΩΣ
        ΓΡΑΨΕ 'ΜΗ ΚΕΝΤΡΟΣΥΜΜΕΤΡΙΚΟΣ ΠΙΝΑΚΑΣ'

    ΑΝ SYM=false ΤΟΤΕ //αντίστροφος
    ΓΙΑ i ← 1 ΜΕΧΡΙ N-1
        ΓΙΑ j ← i+1 ΜΕΧΡΙ N ΑΡΧΗ
            temp←A[i,j]
            A[i,j] ←A[j,i]
            A[j,i] ←temp
    ΤΕΛΟΣ
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
        ΓΙΑ j ← 1 ΜΕΧΡΙ N
            ΓΡΑΨΕ A[i, j]
ΤΕΛΟΣ
```

```
            temp=a[i][j];
            a[i][j]=a[j][i];
            a[j][i]=temp;
        }

  for(i=1;i<=N;i++){
    for(j=1;j<=N;j++)
      printf("%d  ",a[i][j]);
    printf("\n");
  }
  getchar();
}
```

Παρατηρήσεις:

Όπως ήδη έχει γίνει φανερό, πολλές φορές η λύση σε ένα πρόβλημα μπορεί να προκύψει με διαφορετικούς τρόπους. Γενικά, φροντίζουμε πρώτα την ορθότητα της λύσης και στη συνέχεια για την όσο το δυνατόν γρηγορότερη εκτέλεση. Ως προς το δεύτερο σκέλος, είναι προφανές ότι ένας αλγόριθμος που κάνει λιγότερες επαναλήψεις θα ολοκληρωθεί και σε λιγότερο χρόνο εκτέλεσης.

Στον υπολογισμό του αθροίσματος και του γινομένου, θα μπορούσαμε να χρησιμοποιήσουμε εμφωλευμένη επανάληψη με έναν εσωτερικό έλεγχο (ΕΑΝ i=j ή ΕΑΝ j=N-i+1 αντίστοιχα. Ωστόσο, μια τέτοια λύση θα απαιτούσε N*N επαναλήψεις, ενώ στη δική μας περίπτωση απαιτούνται μόλις N επαναλήψεις.

Στον έλεγχο της συμμετρικότητας όπως και στον έλεγχο του άνω τριγωνικού πίνακα, ελέγχονται μόνο τα απολύτως απαραίτητα στοιχεία, ενώ στον έλεγχο του κάτω τριγωνικού πίνακα, αλλά και του κεντροσυμμετρικού πίνακα, για λόγους παραδείγματος, ελέγχονται όλα τα στοιχεία.

Ειδικά για τον κεντροσυμμετρικό πίνακα, εάν απαιτείται ένας γρήγορος αλγόριθμος, θα πρέπει να φροντίσουμε έτσι ώστε να ελεγχθούν μόνο τα απαραίτητα ζεύγη. Ωστόσο, εδώ χρειάζεται προσοχή, καθώς άλλα είναι αυτά τα ζεύγη όταν το N είναι περιττό και άλλα είναι τα ζεύγη όταν το N είναι άρτιο, κάτι που επιβεβαιώνεται από το Σχήμα 9.4. Στη συνέχεια, παρουσιάζεται εναλλακτικά και αυτό το τμήμα αλγορίθμου, το οποίο θα κάνει σε κάθε περίπτωση ακριβώς τις μισές επαναλήψεις από αυτές του αλγορίθμου της λύσης:

```
.....
KENTR←true
AN N mod 2=0 TOTE //άρτιο N
    ΓΙΑ i ← 1 ΜΕΧΡΙ N div 2//ελέγχονται μόνο τα μισά στοιχεία
        ΓΙΑ j ← 1 ΜΕΧΡΙ N
            AN A[i,j]<>A[N-i+1,N-j+1] TOTE
                KENTR←false
ΑΛΛΙΩΣ ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ N div 2 //ελέγχονται μόνο τα μισά στοιχεία πλην της μεσαίας γραμμής
        ΓΙΑ j ← 1 ΜΕΧΡΙ N
            AN A[i,j]<>A[N-i+1,N-j+1] TOTE
                KENTR←false
    ΜΕΣΗ← (N div 2 )+1
    ΓΙΑ j←1 ΜΕΧΡΙ N div 2  //ελέγχονται τα μισά στοιχεία της μεσαίας γραμμής
         AN A[ΜΕΣΗ,j]<>A[ΜΕΣΗ,N-j+1] TOTE
            KENTR←false
ΤΕΛΟΣ //του ΕΑΝ…
AN KENTR=true TOTE
…….
```

Η πλήρης κατανόηση του αλγόριθμου δεν είναι εύκολη μελετώντας απλώς τον κώδικα. Είναι απαραίτητη η χρήση αριθμητικών παραδειγμάτων που θα ελέγχουν όλες τις πιθανές περιπτώσεις. Έτσι, για τον έλεγχο αυτού του αλγόριθμου, αρχικά θα έπρεπε να επιλεχθεί ένα άρτιο Ν (έστω 4), να ελεγχθεί η λειτουργία του αλγόριθμου σε έναν κεντροσυμμετρικό πίνακα αλλά και σε έναν μη κεντροσυμμετρικό πίνακα με γνωστά δεδομένα και στη συνέχεια να επιλεχθεί ένα περιττό Ν (έστω 5) και να επαναληφθεί η διαδικασία του ελέγχου.

9.2.2.14 Συμπίεση δισδιάστατου πίνακα ειδικού τύπου

Πρόβλημα:

«Έστω ότι γνωρίζουμε ότι ένας πίνακας A_{NXN} είναι συμμετρικός. Να αποθηκευθούν τα στοιχεία του πίνακα έτσι ώστε να μη σπαταλιέται χώρος αποθήκευσης»

Συζήτηση:

Όπως γνωρίζουμε, τα στοιχεία ενός συμμετρικού πίνακα είναι συμμετρικώς ίσα ως προς την κύρια διαγώνιο. Άρα, εφόσον γνωρίζουμε ότι ένας τέτοιος πίνακας είναι συμμετρικός, τότε μπορούμε να αποθηκεύσουμε μόνο τα στοιχεία που βρίσκονται κάτω από την κύρια διαγώνιο, μαζί με τα στοιχεία της κυρίας διαγωνίου. Εάν χρειαστεί να ανακατασκευάσουμε τον αρχικό πίνακα, τότε μπορούμε εύκολα να το κάνουμε, λόγω της γνωστής ισότητας των στοιχείων μεταξύ τους. Με αυτόν τον τρόπο θα επιτύχουμε ένα είδος «**συμπίεσης**» (*compression*) των δεδομένων μας.

Τα στοιχεία που θα αποθηκευθούν με αυτόν τον τρόπο είναι αρκετά λιγότερα. Το πλήθος τους υπολογίζεται εύκολα, έχοντας υπόψη την ακόλουθη λογική: Από την $1^{η}$ γραμμή του πίνακα θα αποθηκεύσουμε ένα στοιχείο. Από τη δεύτερη γραμμή θα αποθηκεύσουμε τα στοιχεία της $1^{ης}$ και $2^{ης}$ στήλης. Συνεχίζοντας με αυτόν τον τρόπο, από την Ν-1 γραμμή θα αποθηκευτούν Ν-1 στοιχεία, ενώ από την Ν-στη γραμμή θα αποθηκευτούν Ν στοιχεία. Συνολικά λοιπόν, αντί για N^2 στοιχεία, θα αποθηκευτούν 1+2+3+…+(N-1)+N = N*(N+1)/2 στοιχεία. Έτσι, εάν Ν=4, θα αποθηκευτούν 4*5/2=10 στοιχεία (αντί για 16), αν Ν=5 θα αποθηκευτούν 5*6/2=15 στοιχεία (αντί για 25) κ.ο.κ. Το κέρδος είναι προφανές!

Για να αποθηκεύσουμε τα «χρήσιμα» στοιχεία του αρχικού πίνακα (έστω Α), μπορούμε να χρησιμοποιήσουμε έναν μονοδιάστατο πίνακα (έστω Β), μεγέθους N*(N+1)/2. Τα στοιχεία του Α θα τοποθετηθούν στον Β γραμμή προς γραμμή, επιτυγχάνοντας έτσι την επιθυμητή συμπίεση. Η διαδικασία αυτή φαίνεται με ένα αριθμητικό παράδειγμα στο ακόλουθο σχήμα, για N=5:

	1	*2*	*3*	*4*	*5*
1	**2**	4	1	-1	5
2	**4**	**3**	9	2	4
3	**1**	**9**	**0**	2	3
4	**-1**	**2**	**2**	**7**	1
5	**5**	**4**	**4**	**3**	**2**

Πίνακας Α

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
2	4	3	1	9	0	-1	2	2	7	5	4	4	3	2

Πίνακας Β

Η προσπέλαση των στοιχείων γίνεται πλέον μέσω του πίνακα Β. Έτσι, θα πρέπει να υπάρχει ένας άμεσος τρόπος για την ενημέρωση του κατάλληλου στοιχείου του Β, όταν δίνονται δύο συγκεκριμένα I και J του πίνακα Α. Αντίστροφα, όταν δίνεται ένα K που αντιστοιχεί σε στοιχείο του πίνακα Β, θα πρέπει να υπάρχει ένας άμεσος τρόπος αντιστοίχισής του με τα στοιχεία του πίνακα Α. Στην πρώτη περίπτωση, κάθε στοιχείο A[i,j], με i>=j, πριν το στοιχείο k έχουν αποθηκευτεί όλα τα στοιχεία των προηγούμενων (i-1) γραμμών και τα j στοιχεία της συγκεκριμένης στήλης. Άρα k=1+2+…+i-1+j = (i-1)*i/2+j. Για παράδειγμα, αν i=4 και j=3, το στοιχείο A[4,3] θα αποθηκευτεί στη θέση k=(4-1)*4/2+3=9. Ο τύπος αυτός μπορεί να χρησιμοποιηθεί εναλλακτικά και για τη συμπλήρωση του πίνακα Β.

Σε κάθε περίπτωση συμπίεσης, απαιτείται φυσικά και η αντίστροφη διαδικασία, δηλαδή η αποσυμπίεση. Έτσι, είναι απαραίτητο να ορίσουμε αλγοριθμικά και τον τρόπο με τον οποίο από έναν μονοδιάστατο πίνακα κατάλληλου μεγέθους, θα προκύψει ένας συμμετρικός δισδιάστατος πίνακας.

Τέλος, σημειώνεται ότι η τεχνική που περιγράφεται μπορεί να χρησιμοποιηθεί σε όλες τις περιπτώσεις πινάκων με κάποια ιδιαίτερη ιδιότητα ισότητας στοιχείων. Έτσι, με ακριβώς ίδιο τρόπο μπορούμε να αποθηκεύσουμε έναν άνω τριγωνικό πίνακα, ενώ με μικρές παραλλαγές μπορούμε να διαχειριστούμε τη συμπίεση άλλων τύπων πινάκων (πχ του κεντροσυμμετρικού πίνακα). Σε κάθε περίπτωση, υπολογίζεται πρώτα το πλήθος των διαφορετικών στοιχείων που πρέπει να αποθηκευτούν, ο τρόπος αποθήκευσής τους, όπως και ένας τρόπος αναδημιουργίας του αρχικού πίνακα (αποσυμπίεση).

Η τεχνική αυτή ή παρόμοιά της, δεν μπορεί να χρησιμοποιηθεί σε πίνακες που **δεν** εμφανίζουν κάποια ιδιαίτερη ιδιότητα ισότητας των στοιχείων τους. Σε τέτοιες περιπτώσεις, μπορεί να χρησιμοποιηθεί κάποιος άλλος τρόπος συμπίεσης με τη χρήση

δυναμικών δομών δεδομένων, καθώς είναι αδύνατον να γνωρίζουμε το πλήθος των στοιχείων που θα αποθηκευτούν.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΣΥΜΠ_ΑΠΟΣΥΜΠ
ΣΤΑΘΕΡΕΣ
  N=5
  M=15
ΜΕΤΑΒΛΗΤΕΣ
     A[N,N], B[M], i, j, k: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          ΓΙΑ j ← i ΜΕΧΡΙ N ΑΡΧΗ
               ΔΙΑΒΑΣΕ(A[i, j])
               A[j,i]←A[i,j]
           ΤΕΛΟΣ //είσοδος συμμετρικού πίνακα
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          ΓΙΑ j ← 1 ΜΕΧΡΙ N
               ΓΡΑΨΕ (A[i, j])

    k←1
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          ΓΙΑ j ← 1 ΜΕΧΡΙ i ΑΡΧΗ
               B[k]←A[i, j]
               k←k+1
  //εναλλακτικά: B[(i-1)*i/2+j] ←A[i,j], χωρίς τη χρήση του κ
          ΤΕΛΟΣ // συμπλήρωση συμπιεσμένου πίνακα

    ΓΙΑ i ← 1 ΜΕΧΡΙ N*(N+1)/2
          ΓΡΑΨΕ B[i]

    k←1
    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          ΓΙΑ j ← 1 ΜΕΧΡΙ i ΑΡΧΗ
               A[i, j]←B[k]
               A[j,i]←A[i,j]
               k←k+1
           ΤΕΛΟΣ // αποσυμπίεση πίνακα

    ΓΙΑ i ← 1 ΜΕΧΡΙ N
          ΓΙΑ j ← 1 ΜΕΧΡΙ N
               ΓΡΑΨΕ (A[i, j])

ΤΕΛΟΣ
```

```
#include <stdio.h>
#define N 5
#define M 15
int main(void){
   int i, j, k, a[N+1][N+1], b[M];
  for(i=1;i<=N;i++)
      for(j=1;j<=N;j++){
         scanf("%d", &a[i][j]);fflush(stdin);}

  for(i=1;i<=N;i++){
    for(j=1;j<=N;j++)
      printf("%d   ",a[i][j]);
    printf("\n");
  }

  k=1;
  for(i=1;i<=N;i++)
     for(j=1;j<=i;j++){
        b[k]=a[i][j];
        k++;
        //b[(i-1)*i/2+j]=a[i][j];
}
    printf("\n");
   for(i=1;i<=M;i++)
        printf("%d   ",b[i]);
printf("\n");
  k=1;
  for(i=1;i<=N;i++)
     for(j=1;j<=i;j++){
        a[i][j]=b[k];
        a[j][i]=a[i][j];
        k++;
       }
   for(i=1;i<=N;i++){
     for(j=1;j<=N;j++)
       printf("%d   ",a[i][j]);
     printf("\n");
   }
   getchar();
}
```

9.3 Δυναμικές δομές δεδομένων

Όπως ήδη αναφέρθηκε, οι δυναμικές δομές δεδομένων διαφέρουν από τις στατικές δομές σε δύο βασικά σημεία: στο μέγεθός τους (το οποίο μπορεί να αυξομειώνεται) και στον τρόπο με τον οποίο αποθηκεύονται οι κόμβοι στη μνήμη (στη σειρά στις στατικές δομές, τυχαία στις δυναμικές δομές).

Πριν προχωρήσουμε στην παρουσίαση τέτοιων δομών είναι απαραίτητο να οριστούν δύο νέες έννοιες: η δυνατότητα δημιουργίας νέων τύπων δεδομένων και ο τρόπος με τον οποίο μπορούμε να προσπελάσουμε όχι την τιμή αλλά τη διεύθυνση μιας μεταβλητής, το χώρο δηλαδή μέσα στην κύρια μνήμη στον οποίο φυλάσσεται η τιμή της μεταβλητής.

Μετά την παρουσίαση αυτών των εννοιών, μπορούμε να ασχοληθούμε με περισσότερη προσοχή με τη μελέτη της βασικής δυναμικής δομής δεδομένων, τη

συνδεδεμένη λίστα. Η δομή αυτή είναι ιδιαίτερα σημαντική, καθώς μπορεί να χρησιμοποιηθεί για να προσομοιώσει άλλες, περισσότερο περίπλοκες, δομές δεδομένων που αντιστοιχούν σε κάποιου είδους μοντελοποίησης μιας πραγματικής κατάστασης.

9.3.1 Νέοι τύποι δεδομένων – δομές ή εγγραφές

Όπως ήδη έχει περιγραφεί στο προηγούμενο κεφάλαιο, ο τύπος μιας μεταβλητής καθορίζει τη μορφή των τιμών των δεδομένων που αποθηκεύονται σε αυτή τη μεταβλητή. Έτσι, σε μια μεταβλητή που έχει οριστεί ως ΑΚΕΡΑΙΑ, μόνο ακέραιες τιμές μπορούμε να εκχωρήσουμε μόνο ακέραιες τιμές, σε μια ΛΟΓΙΚΗ μεταβλητή αποθηκεύονται μόνο τιμές true ή false κ.ο.κ.

Ωστόσο, είναι πιθανή η περίπτωση κατά την οποία μια οντότητα του πραγματικού κόσμου που προσπαθούμε να μοντελοποιήσουμε και να αποθηκεύσουμε, να περιγράφεται από περισσότερες από μία τιμές, πιθανώς διαφορετικών τύπων δεδομένων. Έτσι, για παράδειγμα, οι συντεταγμένες ενός σημείου στο καρτεσιανό σύστημα καθορίζονται από μία δυάδα τιμών, (x, y), κάθε μία από τις οποίες αντιστοιχεί σε διαφορετικό άξονα. Σε μια άλλη περίπτωση, ένας μαθητής μπορεί να περιγράφεται όχι μόνο από μια μεταβλητή, αλλά από ένα πλήθος μεταβλητών, όπως για παράδειγμα έναν Αριθμό Μητρώου (ακέραιος), ένα Όνομα (χαρακτήρες), ένα Μέσο Όρο (πραγματικός) κλπ.

Εύκολα θα μπορούσε να χρησιμοποιήσει κανείς τρείς μεταβλητές για να περιγράψει έναν μαθητή όπως αυτόν της προηγούμενης παραγράφου. Επιπλέον, εάν έπρεπε να διαχειριζόμαστε περισσότερους από έναν μαθητές, θα μπορούσαμε να χρησιμοποιήσουμε τρεις διαφορετικούς παράλληλους πίνακες διαφορετικού τύπου, έναν για όλους τους αριθμούς μητρώων, έναν για όλα τα ονόματα κ.ο.κ.

Ωστόσο, στις νεότερες γλώσσες προγραμματισμού, για την αποθήκευση και διαχείριση τέτοιων δεδομένων, προτιμάται η δημιουργία ενός νέου τύπου δεδομένων για κάθε διαφορετική περίπτωση. Έτσι, μπορούμε να ορίσουμε έναν τύπο δεδομένων ΣΗΜΕΙΟ ο οποίος περιλαμβάνει δύο πραγματικούς αριθμούς, για να αναπαραστήσουμε τη δυάδα των καρτεσιανών συντεταγμένων. Μπορούμε επίσης να ορίσουμε έναν τύπο δεδομένων ΜΑΘΗΤΗΣ, ο οποίος περιλαμβάνει έναν ακέραιο (για τον αριθμό μητρώου), ένα αλφαριθμητικό (για το όνομα) και έναν πραγματικό (για τον αριθμό μητρώου του μαθητή.

Η ορισμός ενός νέου τύπου δεδομένων ακολουθείται πάντα από τη δήλωση μιας ή περισσότερων μεταβλητών, για τις οποίες μπορούμε να δηλώσουμε ότι ανήκουν σε αυτό τον νέο τύπο δεδομένων. Είναι φανερό ότι ο ορισμός των νέων τύπων δεδομένων πρέπει να προηγείται από κάθε δήλωση μεταβλητής. Οι νέοι τύποι δεδομένων ονομάζονται αλλιώς ΕΓΓΡΑΦΕΣ ή ΔΟΜΕΣ. Κάθε διαφορετικό τμήμα του νέου τύπου (ο αριθμός μητρώου, το όνομα κλπ), ονομάζεται ΠΕΔΙΟ. Έτσι, μια ΕΓΓΡΑΦΗ μπορεί να περιλαμβάνει πολλά ΠΕΔΙΑ. Στον ψευδοκώδικα που χρησιμοποιούμε, η σύνταξη για τον ορισμό ενός νέου τύπου δεδομένων, μαζί με τον ορισμό μιας μεταβλητής, όπως και ένα παράδειγμα όπου ορίζεται ο τύπος ΜΑΘΗΤΗΣ και δύο μεταβλητές αυτού του τύπου, φαίνονται στη συνέχεια:

```
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ <ονομα_εγγραφής> ΑΡΧΗ
        <ονομα_πεδίου1>:<ΤΥΠΟΣ ΠΕΔΙΟΥ1>
        <ονομα_πεδίου1>:<ΤΥΠΟΣ ΠΕΔΙΟΥ1>
        ……..
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   <ονομα_μεταβλητής>:<ονομα_εγγραφής>
```

```
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΜΑΘΗΤΗΣ ΑΡΧΗ
        Α_Μ:ΑΚΕΡΑΙΕΣ
        ΟΝΟΜΑ:ΧΑΡΑΚΤΗΡΕΣ
        Μ_Ο:ΠΡΑΓΜΑΤΙΚΕΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   ΜΑΘ1, ΜΑΘ2:ΜΑΘΗΤΗΣ
```

Όπως είναι φανερό, στο παράδειγμα ορίζονται δύο μεταβλητές, οι ΜΑΘ1 και ΜΑΘ2, του τύπου ΜΑΘΗΤΗΣ. Αυτό σημαίνει ότι και οι δύο μεταβλητές περιλαμβάνουν από τρία πεδία.

Η προσπέλαση κάθε πεδίου μιας εγγραφής γίνεται με τη βοήθεια του τελεστή '.' (τελεία), η οποία διαχωρίζει το όνομα της μεταβλητής από το συγκεκριμένο πεδίο που προσπελαύνουμε. Έτσι, στο διπλανό παράδειγμα, διαβάζουμε τον αριθμό μητρώου για τον ΜΑΘ1, εκχωρούμε ένα όνομα στο μαθητή ΜΑΘ2 και εμφανίζουμε το μέσο όρο του ΜΑΘ1.

```
ΔΙΑΒΑΣΕ ΜΑΘ1.Α_Μ
ΜΑΘ2.ΟΝΟΜΑ←'ΑΒΡΑΑΜ'
ΓΡΑΨΕ ΜΑΘ1.Μ_Ο
```

Από τη στιγμή που ορίζεται ένας νέος τύπος δεδομένων, μπορούμε να ορίσουμε όχι μόνο νέες μεταβλητές αυτού του τύπου αλλά ακόμα και πίνακες, μονοδιάστατους, δισδιάστατους κλπ, που περιέχουν εγγραφές αυτού του τύπου. Έτσι, στο ακόλουθο παράδειγμα, ορίζουμε και εισάγουμε έναν πίνακα Μ, 100 θέσεων. Κάθε θέση περιέχει μία εγγραφή του τύπου ΜΑΘΗΤΗΣ. Αυτό σημαίνει ότι έχουμε αποθηκεύσει σε μία δομή δεδομένων πολλά δεδομένα διαφορετικού τύπου, ομαδοποιημένα σε εγγραφές.

```
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΜΑΘΗΤΗΣ ΑΡΧΗ
            Α_Μ:ΑΚΕΡΑΙΕΣ
            ΟΝΟΜΑ:ΧΑΡΑΚΤΗΡΕΣ
            Μ_Ο:ΠΡΑΓΜΑΤΙΚΕΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   Μ:ΜΑΘΗΤΗΣ[100]
   i:ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
  ΓΙΑ i←1 ΜΕΧΡΙ 100 ΑΡΧΗ
     ΔΙΑΒΑΣΕ Μ[i].Α_Μ
     ΔΙΑΒΑΣΕ Μ[i].ΟΝΟΜΑ
     ΔΙΑΒΑΣΕ Μ[i].Μ_Ο
  ΤΕΛΟΣ
……..
```

9.3.2 Δείκτες και διευθύνσεις

Στις δυναμικές δομές δεδομένων η μνήμη δεσμεύεται και αποδεσμεύεται κατά τη διάρκεια του προγράμματος. Κατά συνέπεια πρέπει να υπάρχει κάποιος τρόπος για τη διαχείριση της μνήμης, έτσι ώστε τουλάχιστον να είναι δυνατή η προσπέλαση μιας διεύθυνσης μνήμης. Μία διεύθυνση μνήμης είναι φυσικά ένας αριθμός που μπορεί να τυπωθεί ή να αλλάξει μέσω μιας καταχώρησης. Επιπλέον, εφόσον προσπελαύνουμε μια διεύθυνση μνήμης, θα πρέπει να υπάρχει ένας τύπος μεταβλητής στον οποίο να είναι δυνατή η καταχώρηση μιας τέτοιας διεύθυνσης.

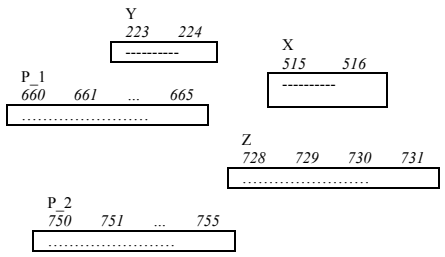
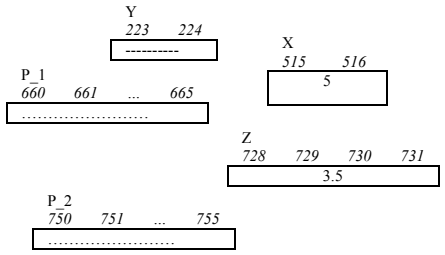
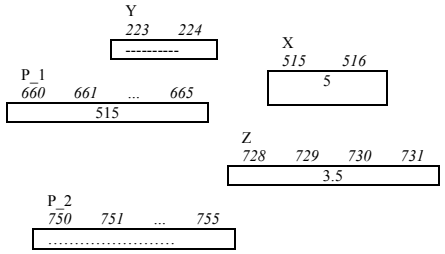
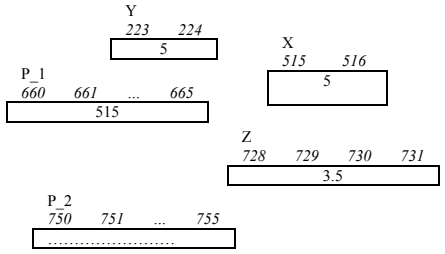
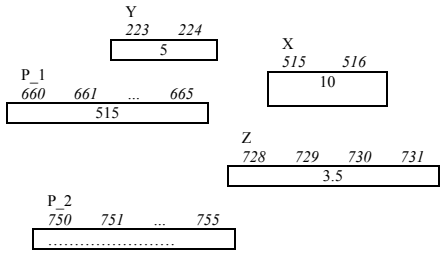
Στον ψευδοκώδικα αυτού του τόμου, προσπελαύνουμε τη διεύθυνση μιας μεταβλητής με τη χρήση του τελεστή **'&'**. Επιπλέον, μία τέτοια διεύθυνση μπορεί να καταχωρηθεί σε μια μεταβλητή τύπου ΔΕΙΚΤΗΣ. Ο τύπος ΔΕΙΚΤΗΣ είναι ο τελευταίος από τους βασικούς τύπους μεταβλητών, μαζί με τους ακέραιους, τους πραγματικούς, τους χαρακτήρες και τους λογικούς. Ωστόσο, η χρήση του είναι αρκετά πιο περίπλοκη από αυτή των άλλων τύπων δεδομένων. Έτσι, δεν αρκεί η δήλωση μιας μεταβλητής τύπου ΔΕΙΚΤΗΣ για να αποθηκεύσει τη διεύθυνση μιας μεταβλητής. Θα πρέπει στη δήλωση αυτή να περιλαμβάνεται και ο τύπος της μεταβλητής, τη διεύθυνση της οποίας αποθηκεύουμε. Ο λόγος είναι ότι διαφορετικού τύπου μεταβλητές δεσμεύουν διαφορετικό μέγεθος μνήμης, άρα απαιτούν κατάλληλο μέγεθος δείκτη! Η δήλωση αυτή γίνεται με τη χρήση του ΔΕΙΚΤΗΣ_ΣΕ <τύπος_μεταβλητής>. Στην πραγματικότητα λοιπόν, ένας δείκτης σε μια μεταβλητή δεν περιέχει ακριβώς τη διεύθυνση μνήμης της μεταβλητής, αλλά τη διεύθυνση μνήμης από την οποία *αρχίζει να αποθηκεύεται* μια μεταβλητή. Η δέσμευση της μνήμης γίνεται κατά τη δήλωση όλων των μεταβλητών και των δεικτών και γίνεται από το μεταγλωττιστή. Σε αυτή τη διαδικασία ο προγραμματιστής δεν έχει καμία αρμοδιότητα.

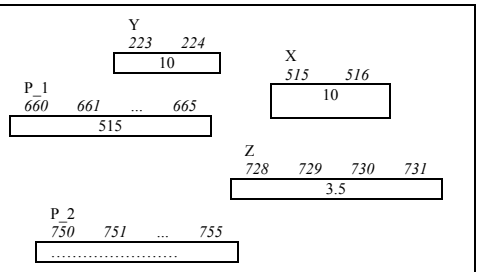
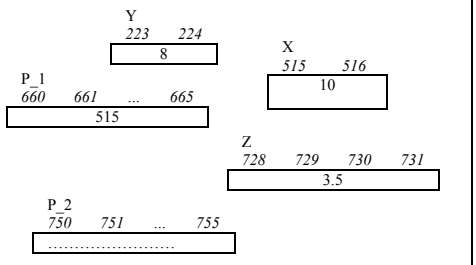
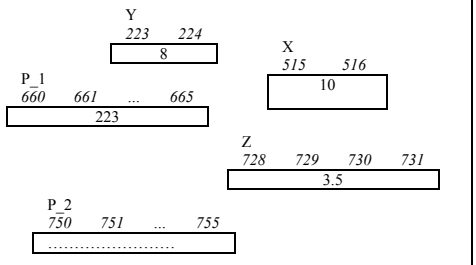
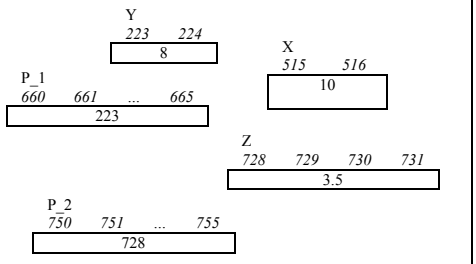
Για να περιπλέξουμε ακόμη περισσότερο τα πράγματα (!), θα χρησιμοποιήσουμε ακόμη έναν τελεστή, τον **'→'**, ο οποίος εφαρμόζεται στο δεξί μέλος μιας μεταβλητής τύπου δείκτη. Ο τελεστής αυτός παράγει την τιμή της διεύθυνσης μνήμης την οποία περιέχει ο δείκτης.

Παρά τη φανερή περιπλοκότητα, στην πράξη η χρήση των δεικτών και των διευθύνσεων μπορεί εύκολα να κατακτηθεί, με την προϋπόθεση της προσοχής στη χρήση τους, όπως και την απόλυτη κατανόηση του τρόπου λειτουργίας τους. Για το λόγο αυτό, θα χρησιμοποιήσουμε το ακόλουθο εκτεταμένο παράδειγμα, στο οποίο για λόγους καλύτερης παρακολούθησης αριθμούμε τις γραμμές:

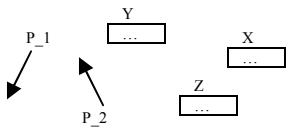
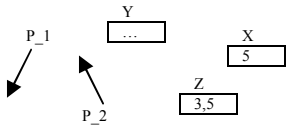
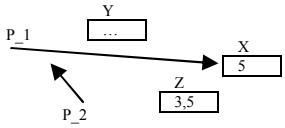
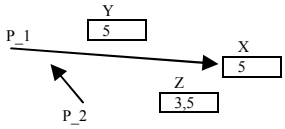
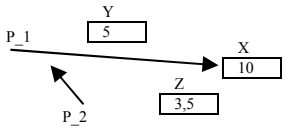
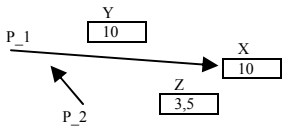
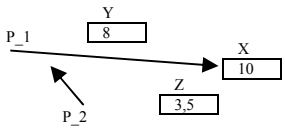
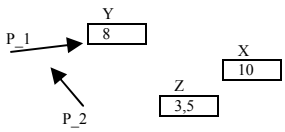
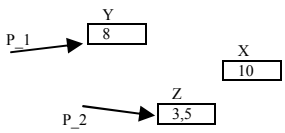
```
ΑΛΓΟΡΙΘΜΟΣ ΠΑΡΑΔΕΙΓΜΑ_ΔΕΙΚΤΩΝ
ΜΕΤΑΒΛΗΤΕΣ                  //δέσμευση χώρου για τα X,Y,Z, P_1, P_2.
1.  X,Y: ΑΚΕΡΑΙΕΣ           // Έστω ότι για το X δεσμεύεται χώρος που ξεκινά από τη  διεύθυνση 515
                             // Έστω ότι για το Y δεσμεύεται χώρος που ξεκινά από τη  διεύθυνση 223
2.  Z:ΠΡΑΓΜΑΤΙΚΕΣ           // Έστω ότι για το  δεσμεύεται χώρος που ξεκινά από τη  διεύθυνση 728
3.  P_1:ΔΕΙΚΤΗΣ_ΣΕ ΑΚΕΡΑΙΕΣ  // Έστω ότι για το P_1 δεσμεύεται χώρος που ξεκινά από τη  διεύθυνση 660
4.  P_2:ΔΕΙΚΤΗΣ_ΣΕ ΠΡΑΓΜΑΤΙΚΕΣ // Έστω ότι για το P_2 δεσμεύεται χώρος που ξεκινά από τη  διεύθυνση 750
ΑΡΧΗ
5.  ΔΙΑΒΑΣΕ X   //έστω X=5
6.  ΔΙΑΒΑΣΕ Z   //έστω Z=3.5
7.  P_1 ← &X    //Στον δείκτη P_1 καταχωρείται η τιμή 515 (προκύπτει από τον τελεστή '&' αριστερά του X
8.  ΓΡΑΨΕ X,Z   //Τυπώνονται τα 5 και 3.5
9.  ΓΡΑΨΕ &X    //Τυπώνεται η τιμή 515
10. ΓΡΑΨΕ P_1   //Τυπώνεται η τιμή 515
11. ΓΡΑΨΕ P_1→          //Τυπώνεται το περιεχόμενο της θέσης που περιέχει ο P_1. Η θέση αυτή είναι η
                        // 515, άρα το περιεχόμενό της είναι το 5 το οποίο και θα τυπωθεί.
12. Y←X          //Στο Y εκχωρείται το 5
13. ΓΡΑΨΕ Y      //Τυπώνεται το 5
14. ΔΙΑΒΑΣΕ X    //Έστω X=10
15. Y←P_1        //Επιχειρείται καταχώρηση δείκτη σε ακέραιο. Η εντολή θα βγάλει ΣΦΑΛΜΑ
16. Y←&X         // Επιχειρείται καταχώρηση διεύθυνσης σε ακέραιο. Η εντολή θα βγάλει ΣΦΑΛΜΑ
17. Y← P_1→      //Στο Y καταχωρείται το περιεχόμενο της θέσης που περιέχει ο P_1. Άρα στο Y θα καταχωρηθεί
                 //η τιμή 10
18. ΓΡΑΨΕ Y      //Τυπώνεται το 10
19. ΔΙΑΒΑΣΕ Y    //Έστω Y=8
20. P_1 = &Y     //Στον P_1 εκχωρείται η διεύθυνση 223
21. P_2←&Y       //ΣΦΑΛΜΑ. Επιχειρείται η εκχώρηση διεύθυνσης ακεραίου σε δείκτη προς πραγματικό
22. P_2←&Z       //Στο P_2 καταχωρείται το 728 (η διεύθυνση του Z)
23. ΓΡΑΨΕ P_2→                  //Τυπώνεται το περιεχόμενο της θέσης που περιέχει ο P_2, δηλαδή το 3.5
24. ΓΡΑΨΕ P_1→                  //Τυπώνεται το 8
ΤΕΛΟΣ
```

Με στόχο την καλύτερη κατανόηση του τρόπου εκτέλεσης του αλγόριθμου, παρουσιάζεται στη συνέχεια μία προσομοίωση της κατάστασης της μνήμης, μαζί με τις αλλαγές που συμβαίνουν σε κάθε εντολή. Προσέξτε ότι για κάθε μεταβλητή, φαίνεται σχηματικά και μία ένδειξη του χώρου που δεσμεύεται. Έτσι, για τις ακέραιες μεταβλητές δεσμεύονται δύο θέσεις μνήμης, για την πραγματική μεταβλητή δεσμεύονται τέσσερις θέσεις μνήμης και για τους δείκτες δεσμεύονται 6 θέσεις μνήμης. Οι ποσότητες αυτές είναι ενδεικτικές, καθώς στις πραγματικές γλώσσες προγραμματισμού εξαρτώνται από την υλοποίηση του μεταγλωττιστή.

Δηλώσεις μεταβλητών, δέσμευση χώρου	
Εντολές 5 και 6	
Εντολή 7	
Εντολή 12	
Εντολή 14	

Εντολή 17	
Εντολή 19	
Εντολή 20	
Εντολή 22	

Τα σχήματα που περιγράφηκαν, μπορούν να απλοποιηθούν ακόμη περισσότερο. Έτσι, δεν χρειάζεται να αναφέρεται η διεύθυνση στην οποία αποθηκεύεται ένας δείκτης. Επιπλέον, εφόσον κάθε δείκτης περιέχει τη διεύθυνση μιας μεταβλητής, μπορεί να από ένα βέλος που «δείχνει σε αυτή τη μεταβλητή». Όταν η τιμή ενός δείκτη αλλάζει, τότε απλώς το βέλος «δείχνει» προς τη νέα διεύθυνση. Στο σχήμα που ακολουθεί, φαίνεται η εκτέλεση του αλγόριθμου με τη χρήση βελών για την αναπαράσταση των δεικτών.

Δηλώσεις μεταβλητών, δέσμευση χώρου (στην αρχή, οι δείκτες δεν δείχνουν πουθενά)	
Εντολές 5 και 6	
Εντολή 7	
Εντολή 12	
Εντολή 14	
Εντολή 17	
Εντολή 19	
Εντολή 20	
Εντολή 22	

Τέλος, για λόγους πληρότητας, παρουσιάζεται και το πρόγραμμα σε C που υλοποιεί τον αλγόριθμο:

```
#include <stdio.h>
int x, y;
float z;
int *p_1;
float *p_2;
int main(void){
   scanf("%d", &x);fflush(stdin);
   scanf("%f", &z);fflush(stdin);
   p_1=&x;
   printf("x=%d z=%f\n",x,z);
   printf("address of x is:%x\n",&x);
   printf("p_1 includes address: %x\n",&x);
   printf("p_1 points to : %d\n",*p_1);
   y=x;
   printf("y is: %d\n",y);
   scanf("%d", &x);fflush(stdin);
   //y=p_1;
   //y=&x;
   y=*p_1;
   printf("y is now: %d\n",y);
   scanf("%d", &y);fflush(stdin);
   p_1=&y;
   //p_2=&y;
   p_2=&z;
   printf("p_2 points to: %f\n",*p_2);
   printf("p_1 points to: %d\n",*p_1);
   getchar();
}
```

9.3.3 Δείκτες σε δομές, αυτοαναφορικές δομές

Ένας δείκτης μπορεί να «δείχνει» μεταβλητή οποιουδήποτε από τους βασικούς τύπους δεδομένων. Ωστόσο, ένας δείκτης μπορεί να δείχνει και σε μία εγγραφή. Στο παράδειγμα που ακολουθεί, χρησιμοποιώντας την εγγραφή του μαθητή που ήδη έχουμε δηλώσει, δηλώνουμε μία μεταβλητή - δείκτη που θα «δείχνει» σε μια τέτοια εγγραφή. Θα πρέπει να προσεχθεί η χρήση του τελεστή '→' στην εντολή «**ΓΡΑΨΕ** P→ONOMA», ο οποίος στη συγκεκριμένη περίπτωση προσπελαύνει το περιεχόμενο του πεδίου ΟΝΟΜΑ της εγγραφής στην οποία δείχνει ο p.

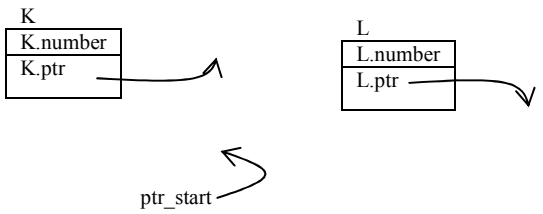
```
ΑΛΓΟΡΙΘΜΟΣ ΔΕΙΚΤΗΣ_ΔΟΜΗ
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΜΑΘΗΤΗΣ ΑΡΧΗ
         Α_Μ:ΑΚΕΡΑΙΕΣ
         ΟΝΟΜΑ:ΧΑΡΑΚΤΗΡΕΣ
         Μ_Ο:ΠΡΑΓΜΑΤΙΚΕΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   Μ:ΜΑΘΗΤΗΣ
   Ρ:ΔΕΙΚΤΗΣ_ΣΕ ΜΑΘΗΤΗΣ
ΑΡΧΗ
    ΔΙΑΒΑΣΕ Μ.Α_Μ
    ΔΙΑΒΑΣΕ Μ.ΟΝΟΜΑ
    ΔΙΑΒΑΣΕ Μ.Μ_Ο
    Ρ←&Μ //ο Ρ «δείχνει» σε ολόκληρη την εγγραφή Μ
   ΓΡΑΨΕ Ρ→ΟΝΟΜΑ
ΤΕΛΟΣ
```

```
#include <stdio.h>
typedef struct mathitis{
     int a_m;
     char onoma[30];
     float m_o;
};
struct mathitis m;
struct mathitis *p;
int main(void){
  scanf("%d",&m.a_m);fflush(stdin);
  scanf("%s",m.onoma);fflush(stdin);
  scanf("%f",&m.m_o);fflush(stdin);
  p=&m;
  printf("%s",p->onoma);
 getchar();
}
```

Ιδιαίτερη ωστόσο σημασία και χρησιμότητα στις δυναμικές δομές δεδομένων έχουν οι λεγόμενες αυτοαναφορικές δομές (εγγραφές). Οι δομές αυτές αποτελούνται από μια σειρά από πεδία οποιουδήποτε πλήθους και τύπου και από ένα τουλάχιστον πεδίο το οποίο δηλώνεται ως δείκτης στην ίδια την εγγραφή. Μια τέτοια δήλωση αυτοαναφορικής δομής, με όνομα ΚΟΜΒΟΣ, που περιλαμβάνει ένα πεδίο με όνομα number, ακέραιου τύπου και ένα πεδίο δείκτη στην ίδια τη δομή, φαίνεται στη συνέχεια:

ΑΛΓΟΡΙΘΜΟΣ ΑΥΤΟΑΝ_ΔΟΜΗ **ΤΥΠΟΙ** **ΕΓΓΡΑΦΗ** ΚΟΜΒΟΣ **ΑΡΧΗ** number:**ΑΚΕΡΑΙΕΣ** ptr: **ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΤΕΛΟΣ** **ΜΕΤΑΒΛΗΤΕΣ** K, L:ΚΟΜΒΟΣ ptr_start:**ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΑΡΧΗ** ……	#include <stdio.h> typedef struct komvos{ int number; struct komvos *ptr; }; struct komvos k,l; struct komvos *ptr_start; int main(void){ ……..

Στο σημείο αυτό, για λόγους κατανόησης, θα περιγράψουμε σχηματικά την κατάσταση που επικρατεί στη μνήμη του υπολογιστή, μετά τις δηλώσεις. Θα χρησιμοποιήσουμε ένα ορθογώνιο για κάθε εγγραφή - κόμβο και ένα βέλος για κάθε δείκτη:



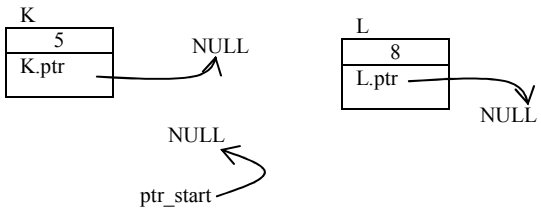
Θα πρέπει να σημειωθεί ότι σε αυτή τη φάση ο απαραίτητος χώρος έχει δεσμευθεί στη μνήμη. Τα πεδία '.number' δεν περιέχουν καμία τιμή ή περιέχουν κάτι αυθαίρετο, ενώ οι δείκτες περιέχουν επίσης αυθαίρετες τιμές, για αυτό και δεν «δείχνουν» πουθενά.

Μολονότι μπορούμε να εισάγουμε κάποια τιμή σε ένα πεδίου τύπου ακεραίου, δεν μπορούμε να κάνουμε το ίδιο σε έναν δείκτη. Οι δείκτες λαμβάνουν τιμές μόνο μέσω καταχώρησης. Επιπλέον, θεωρείται «κακή» προγραμματιστική τεχνική να αφήνουμε κάποιους δείκτες χωρίς να δείχνουν πουθενά. Για αυτό το λόγο, θα αρχικοποιήσουμε όλους τους δείκτες σε μία αρχική τιμή, η οποία ονομάζεται **NULL**. Αρχικοποιώντας έναν δείκτη σε NULL έχουμε πλέον ξεφύγει από την αυθαίρετη (και επικίνδυνη) περίπτωση ο δείκτης να δείχνει αυθαίρετα σε ένα κομμάτι της μνήμης που μπορεί να περιέχει χρήσιμα δεδομένα. Στην επόμενη έκδοση του αλγόριθμου, στις τρεις πρώτες γραμμές αρχικοποιούμε τις δείκτες και στις δύο επόμενες εισάγουμε ακεραίους στα πεδία '.number' των κόμβων K και L (5 και 8 αντίστοιχα)

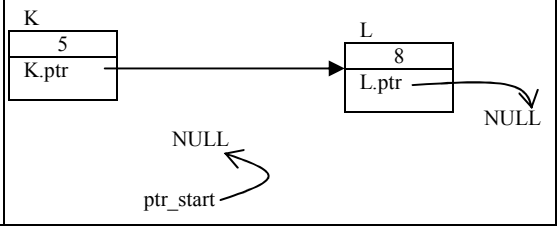
ΑΛΓΟΡΙΘΜΟΣ ΑΥΤΟΑΝ_ΔΟΜΗ **ΤΥΠΟΙ** **ΕΓΓΡΑΦΗ** ΚΟΜΒΟΣ **ΑΡΧΗ** number:**ΑΚΕΡΑΙΕΣ** ptr: **ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΤΕΛΟΣ**	#include <stdio.h> typedef struct komvos{ int number; struct komvos *ptr; }; struct komvos k,l;

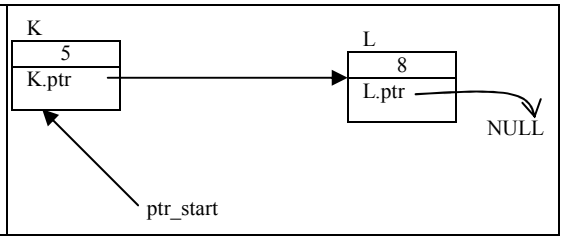
ΜΕΤΑΒΛΗΤΕΣ K, L:ΚΟΜΒΟΣ ptr_start:**ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΑΡΧΗ** K.ptr←**NULL** L.ptr←**NULL** ptr_start ← **NULL** **ΔΙΑΒΑΣΕ** K.number **ΔΙΑΒΑΣΕ** L.number	struct komvos *ptr_start; int main(void){ k.ptr=NULL; l.ptr=NULL; ptr_start=NULL; scanf("%d",&k.number);fflush(stdin); scanf("%d",&l.number);fflush(stdin);

Σχηματικά, η κατάσταση στη μνήμη έχει πλέον ως εξής:



Το σημείο όπου βρισκόμαστε είναι ιδιαίτερα κρίσιμο, καθώς με δύο απλές εντολές μπορούμε πλέον να φτιάξουμε μία λίστα! Πιο συγκεκριμένα, θα εκχωρήσουμε στον K.ptr τη διεύθυνση του κόμβου L (θα βάλουμε τον K.ptr να δείξει στο L) και θα εκχωρήσουμε στον ptr_start τη διεύθυνση του K (θα βάλουμε τον ptr_start να δείξει στον K). Η σειρά με την οποία θα γραφούν οι δύο εντολές δεν έχει σημασία σε αυτή τη φάση.

ΑΛΓΟΡΙΘΜΟΣ ΑΥΤΟΑΝ_ΔΟΜΗ **ΤΥΠΟΙ** **ΕΓΓΡΑΦΗ** ΚΟΜΒΟΣ **ΑΡΧΗ** number:**ΑΚΕΡΑΙΕΣ** ptr: **ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΤΕΛΟΣ** **ΜΕΤΑΒΛΗΤΕΣ** K, L:ΚΟΜΒΟΣ ptr_start:**ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΑΡΧΗ** K.ptr←**NULL** L.ptr←**NULL** ptr_start ← **NULL** **ΔΙΑΒΑΣΕ** K.number **ΔΙΑΒΑΣΕ** L.number	#include <stdio.h> typedef struct komvos{ int number; struct komvos *ptr; }; struct komvos k,l; struct komvos *ptr_start; int main(void){ k.ptr=NULL; l.ptr=NULL; ptr_start=NULL; scanf("%d",&k.number);fflush(stdin); scanf("%d",&l.number);fflush(stdin);
K.ptr ← &L (σε C η εντολή είναι: k.ptr=&l;)	

ptr_start←&K (σε C η εντολή είναι: ptr_start=&k;)	

Σε αυτό ακριβώς το σημείο, έχουμε πετύχει τη σύνδεση των δύο κόμβων και μάλιστα με μια σειρά (ο K είναι πρώτος και ο L είναι δεύτερος) και έχουμε θέσει έναν δείκτη να δείχνει στον πρώτο κόμβο. Πρακτικά, έχουμε επιτύχει τη δημιουργία μιας **λίστας** δύο ακεραίων αριθμών.

Οι δύο τελευταίες εντολές που ακολουθούν θα εκτυπώσουν τα δεδομένα της λίστας (τα περιεχόμενα των πεδίων '.number'), χρησιμοποιώντας μόνο τον δείκτη αρχής της λίστας:

```
ΑΛΓΟΡΙΘΜΟΣ ΑΥΤΟΑΝ_ΔΟΜΗ
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
         number:ΑΚΕΡΑΙΕΣ
         ptr: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   K, L:ΚΟΜΒΟΣ
   ptr_start:ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
ΑΡΧΗ
   K.ptr←NULL
   L.ptr←NULL
   ptr_start ← NULL
   ΔΙΑΒΑΣΕ K.number
   ΔΙΑΒΑΣΕ L.number
   K.ptr ← &L
   ptr_start←&K
   ΓΡΑΨΕ ptr_start→number
   ΓΡΑΨΕ ptr_start→ptr→number
ΤΕΛΟΣ
```

```
#include <stdio.h>
typedef struct komvos{
     int number;
     struct komvos *ptr;
};
struct komvos k,l;
struct komvos *ptr_start;

int main(void){
   k.ptr=NULL;
   l.ptr=NULL;
   ptr_start=NULL;
   scanf("%d",&k.number);fflush(stdin);
   scanf("%d",&l.number);fflush(stdin);
   k.ptr=&l;
   ptr_start=&k;
   printf("%d\n", ptr_start->number);
   printf("%d\n", ptr_start->ptr->number);
   getchar();
}
```

Η δεύτερη εντολή εκτύπωσης εμφανίζει «το περιεχόμενο του πεδίου number του κόμβου στον οποίο δείχνει το πεδίο ptr του κόμβου στον οποίο δείχνει ο δείκτης ptr_start». Φυσικά, είναι ευκολότερο να δεί κανείς την εκτέλεση στο σχετικό σχήμα.

Θα πρέπει βέβαια να τονίσουμε ότι η λίστα που κατασκευάσαμε περιέχει συγκεκριμένο πλήθος κόμβων (2). Κατά συνέπεια, δεν είναι ακόμη μία δυναμική δομή δεδομένων. Στην επόμενη παράγραφο, θα χρησιμοποιήσουμε όσα έχουμε ήδη μάθει για την εισαγωγή στο βασικό εργαλείο των δυναμικών δομών δεδομένων που είναι οι συνδεδεμένες λίστες.

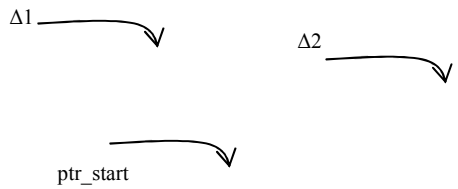
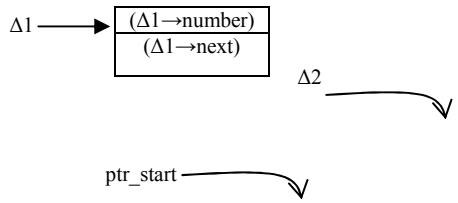
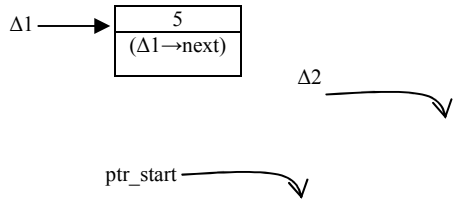
9.3.4 Συνδεδεμένες λίστες

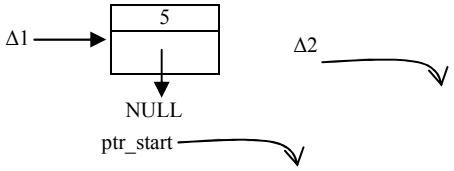
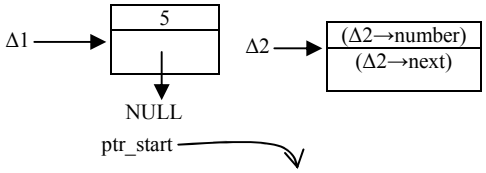
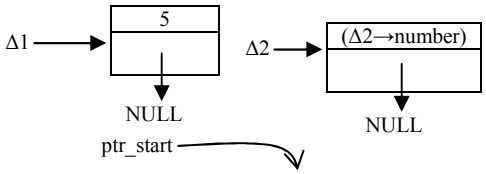
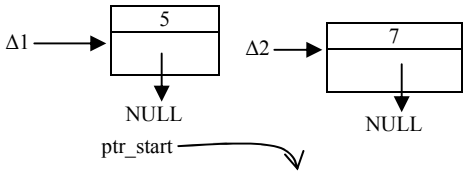
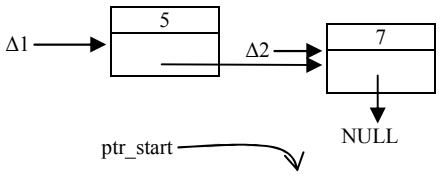
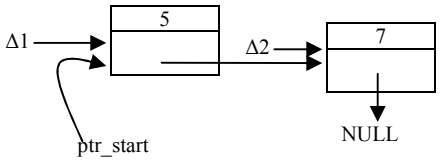
Όπως έχει ήδη αναφερθεί, στις δυναμικές δομές δεδομένων είναι απαραίτητο να υπάρχει ένας τρόπος δυναμικής δέσμευσης μνήμης. Αυτό σημαίνει ότι η απαραίτητη μνήμη για έναν κόμβο της δομής δεδομένων δεν θα γίνει στο τμήμα δηλώσεων (όπως στην προηγούμενη παράγραφο) αλλά κατά τη διάρκεια της εκτέλεσης του προγράμματος. Αυτό με τη σειρά του, σημαίνει ότι είναι απαραίτητη η

χρήση μιας νέας εντολής, η οποία θα δεσμεύει τόσο χώρο, όσο απαιτείται για την αποθήκευση του νέου κόμβου. Στο τμήμα αλγορίθμου που ακολουθεί, φαίνεται η χρήση της εντολής αυτης για τη δημιουργία δύο νέων κόμβων.

ΑΛΓΟΡΙΘΜΟΣ ΔΗΜΙΟΥΡΓΙΑ_ΝΕΩΝ_ΚΟΜΒΩΝ **ΤΥΠΟΙ** **ΕΓΓΡΑΦΗ** ΚΟΜΒΟΣ **ΑΡΧΗ** number:**ΑΚΕΡΑΙΕΣ** next: **ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΤΕΛΟΣ** **ΜΕΤΑΒΛΗΤΕΣ** Δ1, Δ2, ptr_start: **ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΑΡΧΗ** Δ1 ← **ΔΕΣΜΕΥΣΕ_ΧΩΡΟ**(**ΜΕΓΕΘΟΥΣ**(ΚΟΜΒΟΣ)) **ΔΙΑΒΑΣΕ** Δ1→number (Δ1→next) ← **NULL** Δ2 ← **ΔΕΣΜΕΥΣΕ_ΧΩΡΟ**(**ΜΕΓΕΘΟΥΣ**(ΚΟΜΒΟΣ)) (Δ2→next) ← **NULL** **ΔΙΑΒΑΣΕ** Δ2→number (Δ1→next)←Δ2 ptr_start←Δ1 **ΓΡΑΨΕ** ptr_start→number **ΓΡΑΨΕ** ptr_start→next→number **ΤΕΛΟΣ**	#include <stdio.h> #include <stdlib.h> typedef struct komvos{ int number; struct komvos *next; }; struct komvos *d1,*d2, *ptr_start; int main(void){ d1=(struct komvos*)malloc(sizeof(struct komvos)); scanf("%d",&(d1->number));fflush(stdin); d1->next=NULL; d2=(struct komvos*)malloc(sizeof(struct komvos)); scanf("%d",&(d2->number));fflush(stdin); d2->next=NULL; d1->next = d2; ptr_start=d1; printf("%d\n", ptr_start->number); printf("%d\n", ptr_start->next->number); getchar(); }

Η εντολή ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ)), θα δεσμεύσει τον απαραίτητο χώρο στη μνήμη και θα επιστρέψει ως αποτέλεσμα τη διεύθυνση αυτού του χώρου. Η διεύθυνση αυτή καταχωρείται είτε στο δείκτη Δ1 είτε στο δείκτη Δ2. Και πάλι, για λόγους κατανόησης, θα αντιστοιχίσουμε κάθε εντολή με τη σχηματική κατάσταση στη μνήμη.

ΑΛΓΟΡΙΘΜΟΣ ΔΗΜΙΟΥΡΓΙΑ_ΝΕΩΝ_ΚΟΜΒΩΝ **ΤΥΠΟΙ** **ΕΓΓΡΑΦΗ** ΚΟΜΒΟΣ **ΑΡΧΗ** number:**ΑΚΕΡΑΙΕΣ** next: **ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΤΕΛΟΣ** **ΜΕΤΑΒΛΗΤΕΣ** Δ1, Δ2, ptr_start: **ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΑΡΧΗ**	
Δ1 ← **ΔΕΣΜΕΥΣΕ_ΧΩΡΟ**(**ΜΕΓΕΘΟΥΣ**(ΚΟΜΒΟΣ))	
ΔΙΑΒΑΣΕ Δ1→number *//έστω ότι εισάγεται το 5*	

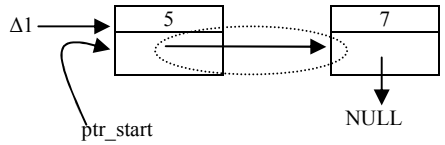
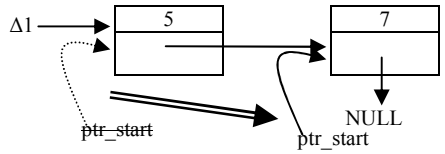
(Δ1→next) ← **NULL**	
Δ2 ← **ΔΕΣΜΕΥΣΕ_ΧΩΡΟ**(**ΜΕΓΕΘΟΥΣ**(ΚΟΜΒΟΣ))	
(Δ2→next) ← **NULL**	
ΔΙΑΒΑΣΕ Δ2→number //έστω ότι εισάγεται το 7	
(Δ1→next)←Δ2 *Με την εντολή αυτή, στο πεδίο δείκτη του 1ου κόμβου (Δ1→next) καταχωρείται το περιεχόμενο του Δ2, δηλαδή η διεύθυνση του 2ου κόμβου. Πρακτικά, το Δ1→next «δείχνει» εκεί που «δείχνει» και ο Δ2*	
ptr_start←Δ1	
ΓΡΑΨΕ ptr_start→number	
ΓΡΑΨΕ ptr_start→next→number **ΤΕΛΟΣ**	

Παρότι και πάλι έχουμε μόλις δύο κόμβους η δομή δεδομένων είναι δυναμική. Βέβαια, και πάλι το πλήθος των κόμβων είναι γνωστό από πριν! Ο στόχος μας, στη συνέχεια, είναι η δημιουργία μιας συνδεδεμένης λίστας για την οποία δεν γνωρίζουμε το πλήθος των κόμβων.

Ωστόσο, πριν ξεκινήσουμε το αλγόριθμο της δημιουργίας μιας συνδεδεμένης λίστας, είναι απαραίτητο να συζητήσουμε τη χρησιμότητα του δείκτη ptr_start που ήδη έχουμε χρησιμοποιήσει. Είναι φανερό ότι ο δείκτης αυτός θα μπορούσε να μην

υπάρχει στον ΔΗΜΙΟΥΡΓΙΑ_ΝΕΩΝ_ΚΟΜΒΩΝ. Έτσι, για την εκτύπωση του περιεχομένου του 1ου κόμβου, θα αρκούσε μια εντολή *ΓΡΑΨΕ Δ1→number*, ενώ για την εκτύπωση των περιεχομένων του 2ου κόμβου μπορούμε να γράψουμε *ΓΡΑΨΕ Δ2→number* ή και *ΓΡΑΨΕ Δ1→next→number*. Τότε όμως, δημιουργείται πρόβλημα στη δομή δεδομένων μας, καθώς δεν υπάρχει ένας μονοσήμαντος τρόπος μέσω του οποίου να προσπελαύνονται τα περιεχόμενά της. Έτσι, για την ίδια δομή δεδομένων θα έπρεπε να αποθηκεύουμε ένα άγνωστο πλήθος δεικτών, κάτι αδύνατο. Για αυτό το λόγο είναι απαραίτητη η αποθήκευση μόνο μιας μεταβλητής, η οποία σηματοδοτεί την έναρξη της δομής μας και μέσω της οποίας γίνεται η προσπέλαση όλης της δομής δεδομένων. Κατά συνέπεια, **ο δείκτης στην αρχή της λίστας είναι απολύτως απαραίτητος!**

Η λύση που χρησιμοποιούμε στο συγκεκριμένο αλγόριθμο προσπελαύνει και τους δύο κόμβους μέσω του ptr_start. Το πρόβλημα όμως είναι ότι εάν οι κόμβοι μας ήταν περισσότεροι από δύο και μάλιστα αγνώστου πλήθους, τότε θα ήταν απίθανο να βρεθεί μια κατάλληλη έκφραση για την προσπέλασή τους. Μία λύση στο πρόβλημα αυτό, θα ήταν να χρησιμοποιήσουμε το δείκτη ptr_start για την προσπέλαση του 1ου κόμβου και στη συνέχεια να «μετακινήσουμε» τον ptr_start έτσι ώστε να «δείχνει» στον επόμενο κατά σειρά κόμβο. Η μετακίνηση αυτή γίνεται εύκολα με μια εντολή καταχώρησης όπως η *ptr_start ← (ptr_start→next)*. Στο διπλανό σχήμα, φαίνεται το αποτέλεσμα μιας τέτοιας καταχώρησης. Μέσα στην έλλειψη φαίνεται ο δείκτης *(ptr_start→next)* **πριν** την εκτέλεση της εντολής. Μετά το τέλος της καταχώρησης η διεύθυνση μνήμης την οποία περιέχει ο *(ptr_start→next)* θα εκχωρηθεί στον *ptr_next*. Κατά συνέπεια, ο ptr_start θα δείχνει πλέον στον επόμενο κόμβο!

Πριν την καταχώρηση:	
Μετά την καταχώρηση: Ο ptr_start «μετακινείται» δεξιά. Η εντολή **ΓΡΑΨΕ** ptr_start→number θα εμφανίσει τώρα το 7	

Παρόλα αυτά τώρα έχουμε δημιουργήσει ένα νέο πρόβλημα: Ο δείκτης αρχής της λίστας έχει μετακινηθεί. Άρα εάν σε κάποιο μεταγενέστερο στάδιο του αλγόριθμου ζητηθεί και πάλι το περιεχόμενο του 1ου κόμβου, θα είναι αδύνατον να το προσπελάσουμε μέσω του δείκτη αρχής. Έτσι, καταλήγουμε σε μια δεύτερη προϋπόθεση σχετικά με το δείκτη στην αρχή της λίστας: **απαγορεύεται η μετακίνησή του!**

Αναγκαστικά λοιπόν, θα πρέπει να χρησιμοποιηθεί μία ακόμη μεταβλητή, τύπου δείκτη, η οποία θα αρχικοποιείται στην αρχή της λίστας και θα «μετακινείται» κάθε φορά που προστίθεται ένας νέος κόμβος στη λίστα, έτσι ώστε να βρίσκεται

πάντα στο τέλος της λίστας. Για το λόγο αυτό στον αλγόριθμο που ακολουθεί χρησιμοποιείται η μεταβλητή current, τύπου δείκτη (τρέχων δείκτης)

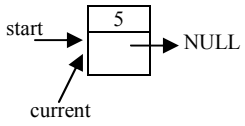
Σε αυτό το σημείο, μπορούμε πλέον να προχωρήσουμε χωρίς προβλήματα κατανόησης. Ο αλγόριθμος που ακολουθεί, «ρωτά» το χρήστη εάν θέλει να προσθέσει έναν νέο κόμβο, και εφόσον η απάντηση είναι θετική, αυτός ο κόμβος θα δημιουργείται, και θα συνδέεται στη λίστα.

```
ΑΛΓΟΡΙΘΜΟΣ ΔΗΜΙΟΥΡΓΙΑ_ΛΙΣΤΑΣ
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
           number:ΑΚΕΡΑΙΕΣ
           next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   start, current, neo: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   C:ΧΑΡΑΚΤΗΡΕΣ
ΑΡΧΗ
  start←NULL
  ΓΡΑΨΕ 'ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ ΚΟΜΒΟΥ?(Ν/Ο)'
  ΔΙΑΒΑΣΕ C
  ΟΣΟ C='Ν' ΑΡΧΗ
     ΑΝ start=NULL ΤΟΤΕ ΑΡΧΗ
        start←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
        (start→next) ←NULL
        ΔΙΑΒΑΣΕ start→number  //έστω 5
        current←start
     ΤΕΛΟΣ
     ΑΛΛΙΩΣ ΑΡΧΗ
        neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
        (neo→next) ←NULL
        ΔΙΑΒΑΣΕ neo→number
        (current→next)←neo
        current←(current→next) //ή current ← neo
     ΤΕΛΟΣ
     ΓΡΑΨΕ 'ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ ΚΟΜΒΟΥ?(Ν/Ο)'
     ΔΙΑΒΑΣΕ C
  ΤΕΛΟΣ //της ΟΣΟ
ΤΕΛΟΣ
```

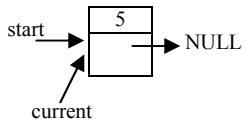
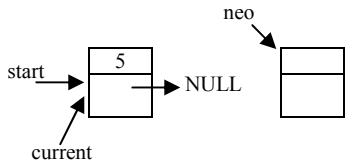
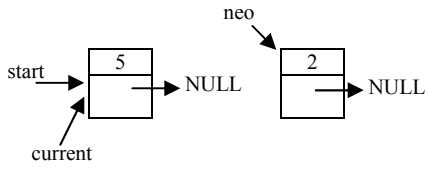
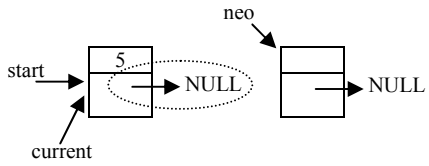
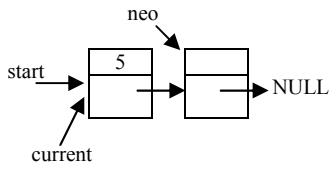
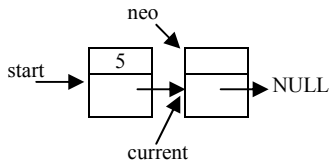
```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
     int number;
     struct komvos *next;
};
struct komvos *start, *current, *neo;
char c;
int main(void){
  start=NULL;
  printf("create new node (y,n)");
  scanf("%c",&c);fflush(stdin);
  while(c=='y'){
   if(start==NULL){
    start=(struct komvos*)malloc(sizeof(struct komvos));
    start->next=NULL;
    scanf("%d",&start->number);fflush(stdin);
    current=start;
   }
   else {
    neo=(struct komvos*)malloc(sizeof(struct komvos));
    neo->next=NULL;
    scanf("%d",&neo->number);fflush(stdin);
    current->next=neo;
    current=neo;
    }
   printf("create new node (y,n)");
   scanf("%c",&c);fflush(stdin);
  }
  getchar();
}
```

Η λειτουργία της επανάληψης είναι απλή. Ωστόσο, μέσα στην επανάληψη, διακρίνουμε δύο περιπτώσεις: είτε η λίστα να είναι κενή (start=NULL) είτε η λίστα να έχει ήδη κάποιους κόμβους.

Στην 1η περίπτωση, δημιουργείται ένας νέος κόμβος, εισάγονται δεδομένα και τοποθετείται ο δείκτης current ώστε να δείχνει σε αυτόν τον κόμβο. Σχηματικά, εφόσον start=NULL, στο τέλος της ομάδας εντολών του ΑΝ…, η κατάσταση θα έχει ως εξής:



Στη δεύτερη περίπτωση, η λίστα έχει ήδη κάποιους κόμβους. Εδώ θα τεθεί σε χρήση και ο δείκτης current. Θα υποθέσουμε ότι έχουμε περάσει στη 2η επανάληψη και ο χρήστης θα προσθέσει έναν κόμβο με τιμή 4 στο πεδίο number.

Έως τώρα:	
neo←**ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ**(ΚΟΜΒΟΣ))	
(neo→next) ←**NULL** **ΔΙΑΒΑΣΕ** neo→number *//έστω 2*	
(current→next)←neo (πριν την καταχώρηση) Ο current→next φαίνεται μέσα σε έλλειψη. Μετά την ολοκλήρωση της καταχώρησης ο δείκτης αυτός θα δείχνει εκεί όπου δείχνει και ο δείκτης neo	
(current→next)←neo (μετά την καταχώρηση)	
current ← neo *//ή current←(current→next)*	

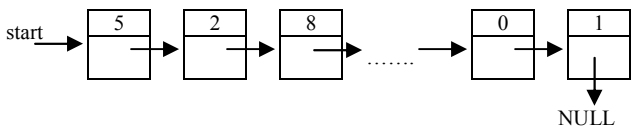
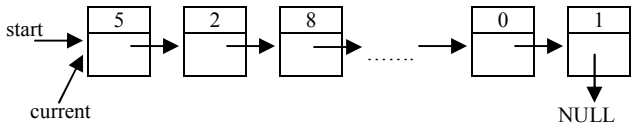
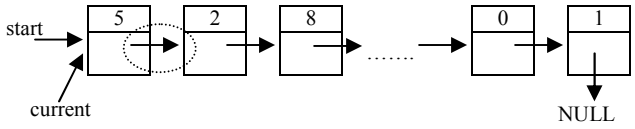
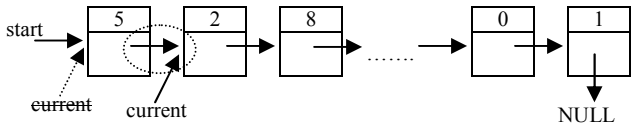
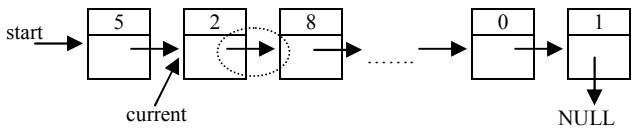
Όπως είναι φανερό, στο τέλος της επανάληψης έχει συνδεθεί ο νέος κόμβος με την υπάρχουσα λίστα και ο δείκτης current δείχνει στον κόμβο αυτόν. Κάθε φορά που η διαδικασία επαναλαμβάνεται, θα δημιουργείται ένας νέος κόμβος, θα συνδέεται μέσω της καταχώρησης (current→next)←neo με το τέλος της υπάρχουσας λίστας και ο δείκτης current θα δείχνει στο καινούργιο τέλος της λίστας. Ο αλγόριθμός μας έχει ολοκληρωθεί!

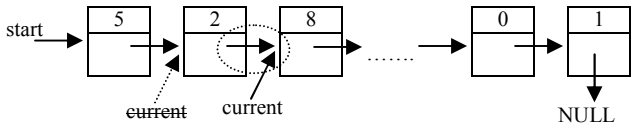
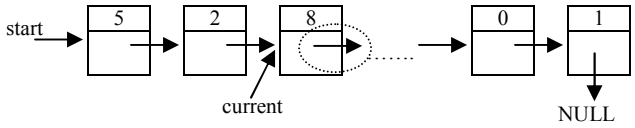
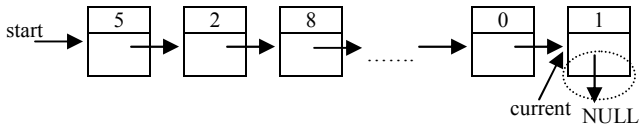
9.3.5 Διαπέραση συνδεδεμένης λίστας

Στην παράγραφο αυτή θα παρατεθεί ένα τμήμα αλγόριθμου το οποίο θα διατρέχει μια ήδη κατασκευασμένη συνδεδεμένη λίστα. Το τμήμα αυτό αλγορίθμου, επισκέπτεται κάθε κόμβο της λίστας, τυπώνει το περιεχόμενο των δεδομένων του και σταματά εντοπίζοντας τον τελευταίο κόμβο στη λίστα. Η χρήση του τμήματος αυτού αλγορίθμου θα χρησιμοποιείται πολύ συχνά στη συνέχεια.

ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΠΕΡΑΣΗ_ΛΙΣΤΑΣ **ΤΥΠΟΙ** **ΕΓΓΡΑΦΗ** ΚΟΜΒΟΣ **ΑΡΧΗ** number:**ΑΚΕΡΑΙΕΣ** next: **ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΤΕΛΟΣ** **ΜΕΤΑΒΛΗΤΕΣ** start, current: **ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ **ΑΡΧΗ** *// Έστω ότι έχει δημιουργηθεί μία συνδεδεμένη λίστα με μη* *// μηδενικό πλήθος κόμβων. Έστω ότι ο δείκτης start «δείχνει»* *// στον πρώτο κόμβο της λίστας και ο δείκτης του τελευταίου* *// κόμβου δείχνει σε NULL* current←start **ΟΣΟ** (current→next)<>**NULL ΑΡΧΗ** **ΓΡΑΨΕ** current→number current ← (current→next) **ΤΕΛΟΣ** **ΓΡΑΨΕ** current→number **……..** **ΤΕΛΟΣ**	#include <stdio.h> #include <stdlib.h> typedef struct komvos{ int number; struct komvos *next; }; struct komvos *start, *current; int main(void){ //……. current = start; while(current->next!=NULL){ printf("%d ", current->number); current = current->next; } printf("%d ", current->number); getchar(); }

Ας δούμε και πάλι σχηματικά την εκτέλεση αυτού του τμήματος αλγορίθμου, εντολή προς εντολή:

Αρχικά:	
current←start	
Ο current δείχνει στον 1ο κόμβο. Έτσι, το current→number είναι το 5 και το current→next σημειώνεται μέσα σε έλλειψη. Προφανώς, current→next <> NULL, άρα εισερχόμαστε στην επανάληψη	
Τυπώνεται το 5. Η επόμενη καταχώρηση θέτει το περιεχόμενο του current→next μέσα στον current. Όπως ήδη έχουμε αναφέρει, μετά την εκτέλεση αυτής της καταχώρησης ο current θα δείχνει εκεί που δείχνει ο current→next. Ουσιαστικά, μετακινήσαμε τον current δεξιά !	
Ο current δείχνει πλέον στο 2ο κόμβο. Έτσι, το current→number είναι το 2 και το current→next σημειώνεται μέσα σε έλλειψη. Προφανώς, current→next <> NULL, άρα εισερχόμαστε στην επανάληψη	

Τυπώνεται το 2. Η επόμενη καταχώρηση θέτει το περιεχόμενο του current→next μέσα στον current.	
Ο current δείχνει στον 3ο κόμβο. Έτσι, το current→number είναι το 8 και το current→next σημειώνεται μέσα σε έλλειψη. Προφανώς, current→next <> NULL, άρα εισερχόμαστε στην επανάληψη.	
Στις επόμενες επαναλήψεις, θα τυπώνεται το περιεχόμενο του τρέχοντος κόμβου (του κόμβου στον οποίο δείχνει ο δείκτης current) και αμέσως μετά ο δείκτης current θα μετακινείται δείχνοντας στον επόμενο κόμβο. Ας υποθέσουμε ότι έχει τυπωθεί το 0 και ότι ο current δείχνει πλέον στον τελευταίο κόμβο. Στην έλλειψη φαίνεται το current→next	

Προφανώς current→next = NULL, άρα θα βγούμε από την επανάληψη. Σε αυτό το σημείο του αλγόριθμου, έχουμε διατρέξει όλη τη λίστα και έχουμε θέσει τον δείκτη current να δείχνει στον τελευταίο κόμβο της. Βέβαια, εφόσον δεν μπήκαμε στην επανάληψη, το περιεχόμενο του τελευταίου κόμβου δεν έχει ακόμα τυπωθεί, για αυτό και υπάρχει η εντολή ΓΡΑΨΕ current→number στο τέλος του αλγόριθμου. Η εντολή αυτή θα εμφανίσει το 1.

```
……
current←start
  ΟΣΟ current<>NULL ΑΡΧΗ
    ΓΡΑΨΕ current→number
    current ← (current→next)
  ΤΕΛΟΣ
……
```

Είναι εύκολο να επιβεβαιώσει κανείς ότι εάν ο στόχος μας ήταν απλώς να διατρέξουμε τη λίστα και να τυπώσουμε όλα τα περιεχόμενα των κόμβων, θα αρκούσαν οι εντολές που φαίνονται παραπλεύρως. Ωστόσο, στο τέλος μιας τέτοιας επανάληψης, ο δείκτης current θα δείχνει σε NULL και όχι στον τελευταίο κόμβο της λίστας. Είναι προφανές ότι χρησιμοποιούμε όποιον τρόπο είναι απαραίτητος, ανάλογα με τα ζητούμενα του αλγόριθμου.

9.3.6 Εφαρμογές στις συνδεδεμένες λίστες

Όπως και στην περίπτωση των πινάκων, η πληρέστερη κατανόηση της λειτουργίας των συνδεδεμένων λιστών θα προκύψει μέσω της ανάπτυξης εκτεταμένων εφαρμογών – παραδειγμάτων. Σε αυτές τις εφαρμογές, συχνά θα θεωρούμε ότι κάποια λίστα έχει ήδη δημιουργηθεί. Συνήθως, πρόκειται για λίστες που αποτελούνται από τον ίδιο τύπο κόμβων που χρησιμοποιήσαμε έως τώρα (ένας ακέραιος και ένας δείκτης). Προκειμένου να αποφύγουμε την επανάληψη γνωστών τμημάτων αλγορίθμων, θα σημειώνουμε λεκτικά μέσα σε σχόλια αυτές τις διαδικασίες. Τα προγράμματα σε C θα πρέπει να συμπεριλαμβάνουν φυσικά όλες τις σχετικές εντολές, κάτι που μπορεί να γίνει με απλή αντιγραφή αυτών των εντολών και ενημέρωση των απαραίτητων μεταβλητών.

Ωστόσο, κρίσιμο είναι να τονιστεί ότι η πλήρης κατανόηση του τρόπου επίλυσης των προβλημάτων γίνεται μόνο με την προσεκτική, βήμα-προς-βήμα εκτέλεση του κάθε αλγόριθμου, συνοδευμένη από τη σχεδίαση των αντίστοιχων

σχημάτων-σκαριφημάτων σε ένα πρόχειρο. Είναι πρακτικά αδύνατον να μπορέσει κάποιος να αντιληφθεί με πληρότητα και επάρκεια τη λειτουργία μιας δυναμικής δομής δεδομένων χωρίς τα κατάλληλα σκαριφήματα.

9.3.6.1 Πράξεις σε συνδεδεμένες λίστες

Πρόβλημα:
«Δίνεται μια συνδεδεμένη λίστα με δείκτη αρχής start και κόμβους που περιέχουν από ένα πεδίο με έναν ακέραιο αριθμό. Να υπολογίσετε το άθροισμα των θετικών αριθμών, το γινόμενο των αρνητικών αριθμών και το μέσο όρο των άρτιων αριθμών που είναι αποθηκευμένοι στη λίστα.»

Συζήτηση:
Οι υπολογισμοί που απαιτούνται θα γίνουν με τον ίδιο ακριβώς τρόπο που έχουν γίνει και στους πίνακες. Η διαφορά είναι ότι τα δεδομένα είναι αποθηκευμένα σε λίστα, οπότε θα πρέπει να διατρέξουμε τη λίστα μας, από την αρχή ως το τέλος. Για κάθε κόμβο που επισκεπτόμαστε, θα ελέγχουμε ανάλογα τον ακέραιο που έχει στο πεδίο number και θα εκτελούμε την κατάλληλη πράξη.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΥΠΟΛΟΓΙΣΜΟΙ_ΛΙΣΤΑΣ
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
           number:ΑΚΕΡΑΙΕΣ
           next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   start, current: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   S, SA, P, ΠΛΗΘΟΣ: ΑΚΕΡΑΙΕΣ
   ΜΟ:ΠΡΑΓΜΑΤΙΚΕΣ
ΑΡΧΗ
// Έστω ότι έχει δημιουργηθεί μία συνδεδεμένη λίστα με μη
// μηδενικό πλήθος κόμβων. Έστω ότι ο δείκτης start «δείχνει»
// στον πρώτο κόμβο της λίστας και ο δείκτης του τελευταίου
// κόμβου δείχνει σε NULL
  S←0
  SA←0
  P←1
  ΠΛΗΘΟΣ ←0
  current←start
  ΟΣΟ current<>NULL ΑΡΧΗ
     ΑΝ current→number>0 ΤΟΤΕ
       S←S+current→number
     ΑΛΛΙΩΣ
       ΑΝ current→number<0 ΤΟΤΕ
         P ← P*current→number
     ΑΝ current→number mod 2 = 0 ΤΟΤΕ ΑΡΧΗ
       SA←SA+current→number
       ΠΛΗΘΟΣ←ΠΛΗΘΟΣ+1
     ΤΕΛΟΣ
     current ← (current→next)
  ΤΕΛΟΣ
  ΓΡΑΨΕ S
  ΓΡΑΨΕ P
  ΑΝ ΠΛΗΘΟΣ>0 ΤΟΤΕ ΑΡΧΗ
    ΜΟ←ΣΑ/ΠΛΗΘΟΣ
    ΓΡΑΨΕ ΜΟ
  ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
     int number;
     struct komvos *next;
};
struct komvos *start, *current;
int  s, p, sa, plithos;
float mo;

int main(void){
   //……list created…….
  s=0;p=1;plithos=0;sa=0;
  current = start;
  while(current!=NULL){
         if(current->number>0)
            s+= current->number;
         else
            if (current->number<0)
               p*= current->number;
         if(current->number%2==0){
           sa+= current->number;
           plithos++;
         }
         current = current->next;
  }
   printf("sum:%d, product:%d\n", s,p);
   if(plithos){
     mo=sa/plithos;
     printf("%4.2f",mo);
   }
   getchar();
}
```

9.3.6.2. Μετατροπή πίνακα σε λίστα

Πρόβλημα:

«Δίνεται μονοδιάστατος πίνακα ακεραίων Α, 10 θέσεων. Να αντιγραφούν όλα τα στοιχεία του σε μία συνδεδεμένη λίστα»

Συζήτηση:

Η εφαρμογή θα επιλυθεί με μία διαπέραση του πίνακα. Για κάθε στοιχείο του, θα δημιουργείται ένας νέος κόμβος και θα συνδέεται στη λίστα.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΠΙΝΑΚΑΣ_ΣΕ_ΛΙΣΤΑ
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
           number:ΑΚΕΡΑΙΕΣ
           next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   start, current, neo: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   A[10], i: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
  ΓΙΑ i←1 ΜΕΧΡΙ 10
     ΔΙΑΒΑΣΕ A[i]
  start←NULL
  ΓΙΑ i←1 ΜΕΧΡΙ 10 ΑΡΧΗ
      ΑΝ start=NULL ΤΟΤΕ ΑΡΧΗ
         start←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
         (start→next) ←NULL
         (start→number)  ← A[1]
         current←start
      ΤΕΛΟΣ
      ΑΛΛΙΩΣ ΑΡΧΗ
         neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
         (neo→next) ←NULL
         (neo→number)←A[i]
         (current→next)←neo
         current ← neo
      ΤΕΛΟΣ
     ΤΕΛΟΣ //της ΓΙΑ…
     current←start
     ΟΣΟ current<>NULL ΑΡΧΗ
        ΓΡΑΨΕ current→number
        current←(current→next)
     ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
      int number;
      struct komvos *next;
};
struct komvos *start, *current, *neo;
int  a[11], i;

int main(void){
    for(i=1;i<=10;i++){
        scanf("%d", &a[i]);fflush(stdin);}
   start=NULL;
   for(i=1;i<=10;i++){
      if(start==NULL){
         start=(struct komvos*)malloc(sizeof(struct komvos));
         start->next=NULL;
         start->number=a[i];
         current=start;
      }
      else {
         neo=(struct komvos*)malloc(sizeof(struct komvos));
         neo->next=NULL;
         neo->number=a[i];
         current->next=neo;
         current=neo;
       }
     }
      current=start;
      while(current!=NULL){
         printf("%d ",  current->number);
         current=current->next;
      }
    getchar();
}
```

9.3.6.3 Αντιγραφή κάποιων στοιχείων πίνακα σε λίστα

Πρόβλημα:

«Δίνεται δισδιάστατος πίνακας ακεραίων Α5Χ5. Να δημιουργηθεί συνδεδεμένη λίστα που να αποτελείται μόνο από τα θετικά στοιχεία του πίνακα»

Συζήτηση:

Η λύση είναι παρόμοια με την προηγούμενη. Διαφέρει η διαπέραση του δισδιάστατου πίνακα αλλά και ο έλεγχος των θετικών στοιχείων του.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΠΙΝΑΚΑΣ_ΣΕ_ΛΙΣΤΑ_1
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
           number:ΑΚΕΡΑΙΕΣ
           next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   start, current, neo: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   A[5][5], i, j: ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
  ΓΙΑ i←1 ΜΕΧΡΙ 5
     ΓΙΑ j←1 ΜΕΧΡΙ 5
     ΔΙΑΒΑΣΕ A[i, j]
  start←NULL
  ΓΙΑ i←1 ΜΕΧΡΙ 5
    ΓΙΑ j←1 ΜΕΧΡΙ 5
     ΑΝ a[i][j]>0 ΤΟΤΕ
      ΑΝ start=NULL ΤΟΤΕ ΑΡΧΗ
         start←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
         (start→next) ←NULL
         (start→number)  ← A[i][j]
         current←start
       ΤΕΛΟΣ
       ΑΛΛΙΩΣ ΑΡΧΗ
         neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
         (neo→next) ←NULL
         (neo→number)←A[i][j]
         (current→next)←neo
         current ← neo
       ΤΕΛΟΣ

     current←start
     ΟΣΟ current<>NULL ΑΡΧΗ
        ΓΡΑΨΕ current→number
        current←(current→next)
     ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
      int number;
      struct komvos *next;
};
struct komvos *start, *current, *neo;
int  a[6][6], i,j;

int main(void){
    for(i=1;i<=5;i++)
        for(j=1;j<=5;j++){
        scanf("%d", &a[i][j]);fflush(stdin);}
   start=NULL;
   for(i=1;i<=5;i++)
     for(j=1;j<=5;j++)
       if(a[i][j]>0)
        if(start==NULL){
          start=(struct komvos*)malloc(sizeof(struct komvos));
          start->next=NULL;
          start->number=a[i][j];
          current=start;
        }
        else {
           neo=(struct komvos*)malloc(sizeof(struct komvos));
           neo->next=NULL;
           neo->number=a[i][j];
           current->next=neo;
           current=neo;
         }

       current=start;
       while(current!=NULL){
          printf("%d ",  current->number);
          current=current->next;
       }
    getchar();
}
```

9.3.6.4 Προσθήκη κόμβου στην αρχή και στο τέλος

Πρόβλημα:

«Δίνεται συνδεδεμένη λίστα ακεραίων με δείκτη αρχής start. Να προστεθεί ένας κόμβος στην αρχή της και ένας κόμβος στο τέλος της»

Συζήτηση:

Η προσθήκη ενός κόμβου στην αρχή μιας λίστας είναι μια πολύ απλή διαδικασία. Δημιουργούμε έναν νέο κόμβο, τον βάζουμε να δείχνει εκεί που δείχνει ο πρώτος κόμβος και θέτουμε την αρχή να δείχνει στον νέο κόμβο.

Η προσθήκη ενός κόμβου στο τέλος της λίστας, προϋποθέτει τη διαπέραση της λίστας και την τοποθέτηση ενός δείκτη στο τέλος της. Στη συνέχεια δημιουργείται ένας κόμβος και συνδέεται με τη λίστα. Σε κάθε περίπτωση, θα πρέπει να ελέγχεται η περίπτωση της κενής λίστας, καθώς οι απαιτούμενες ενέργειες αλλάζουν όταν μια λίστα είναι κενή. Ειδικά για την προσθήκη στο τέλος, όταν η λίστα είναι κενή τότε η διαδικασία είναι ακριβώς ίδια με την προσθήκη στην αρχή.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΥΠΟΛΟΓΙΣΜΟΙ_ΛΙΣΤΑΣ
```

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

```
ΤΥΠΟΙ
    ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
            number:ΑΚΕΡΑΙΕΣ
            next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
    ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
    start, current, neo: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
ΑΡΧΗ
// Έστω ότι έχει δημιουργηθεί μία συνδεδεμένη λίστα με μη
// μηδενικό πλήθος κόμβων. Έστω ότι ο δείκτης start «δείχνει»
// στον πρώτο κόμβο της λίστας και ο δείκτης του τελευταίου
// κόμβου δείχνει σε NULL

   ΑΝ start=NULL ΤΟΤΕ ΑΡΧΗ
     neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
     (neo→next) ←NULL
     ΔΙΑΒΑΣΕ neo→number
     start=neo //πρόσθεση σε κενή λίστα
   ΤΕΛΟΣ
   ΑΛΛΙΩΣ  ΑΡΧΗ  //μη κενή λίστα
        neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
        ΔΙΑΒΑΣΕ neo→number
        (neo→next) ← start
        start←neo //σύνδεση στην αρχή
        //προστέθηκε κόμβος στην αρχή

        current←start
        ΟΣΟ (current→next<>NULL)  //βρες το τέλος
            current←(current→next)
        neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
       (neo→next) ←NULL
       ΔΙΑΒΑΣΕ neo→number
       (current→next) ←neo //σύνδεση στο τέλος
    ΤΕΛΟΣ //μη κενή λίστα
    current←start
    ΟΣΟ current<>NULL ΑΡΧΗ
         ΓΡΑΨΕ current→number
         current←(current→next)
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdlib.h>
typedef struct komvos{
     int number;
     struct komvos *next;
};
struct komvos *start, *current, *neo;

int main(void){
   //……list created…….



  if(start==NULL){
      neo=(struct komvos*)malloc(sizeof(struct komvos));
      neo->next=NULL;
      printf("list empty, give a number:");
      scanf("%d", &neo->number);fflush(stdin);
      start=neo;
      }
    else{
        neo=(struct komvos*)malloc(sizeof(struct komvos));
        printf("list full, give first number:");
        scanf("%d", &neo->number);fflush(stdin);
        neo->next=start;
        start=neo;

        current=start;
        while(current->next!=NULL)
          current=current->next;
        neo=(struct komvos*)malloc(sizeof(struct komvos));
        neo->next=NULL;
        printf("list full, give last number:");
        scanf("%d", &neo->number);fflush(stdin);
        current->next=neo;
      }
  current=start;
  while(current!=NULL){
        printf("%d ",  current->number);
        current=current->next;
   }
getchar();
}
```

9.3.6.5 Προσθήκη κόμβου σε κάποιο αυθαίρετο σημείο

Πρόβλημα:

«Δίνεται συνδεδεμένη λίστα διαφορετικών ακεραίων με δείκτη αρχής start.Είναι γνωστό ότι κάποιος από τους κόμβους περιέχει το 5. Να προστεθεί ένας κόμβος μετά τον κόμβο που περιέχει το 5»

Συζήτηση:

Σε αυτή την περίπτωση είναι σαφές ότι μετά τη δημιουργία του νέου κόμβου, θα πρέπει να βρεθεί η θέση στην οποία θα συνδεθεί. Εφόσον γνωρίζουμε ότι υπάρχει κόμβος με την τιμή 5, θα πρέπει να διατρέξουμε τη λίστα έως ότου βρεθεί αυτός ο κόμβος. Από τη στιγμή που τεθεί ο τρέχων δείκτης στον κόμβο με την τιμή 5, είναι πλέον απλό να συνδέσουμε τον νέο κόμβο ανάμεσα στον τρέχοντα δείκτη και τον επόμενο κόμβο. Τέλος, εφόσον γνωρίζουμε ότι υπάρχει τουλάχιστον ένας κόμβος στη λίστα, δεν είναι απαραίτητος ο έλεγχος κενής λίστας. Ωστόσο, στην περίπτωση που ο

νέος κόμβος τοποθετηθεί στο τέλος της λίστας, θα πρέπει να φροντίσουμε ώστε η νέα λίστα να τερματίζεται κανονικά.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΠΡΟΣΘΗΚΗ_ΚΟΜΒΟΥ
ΤΥΠΟΙ
    ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
            number:ΑΚΕΡΑΙΕΣ
            next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
    ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
    start, current, neo: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
ΑΡΧΗ
// Έστω ότι έχει δημιουργηθεί μία συνδεδεμένη λίστα με μη
// μηδενικό πλήθος κόμβων. Έστω ότι ο δείκτης start «δείχνει»
// στον πρώτο κόμβο της λίστας και ο δείκτης του τελευταίου
// κόμβου δείχνει σε NULL
    neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
    ΔΙΑΒΑΣΕ neo→number
    current←start
   ΟΣΟ (current<>NULL and current→number<>5)
      current←(current→next) //o current δείχνει στο 5
    AN current→next=NULL ΑΡΧΗ  //αν είναι ο τελευταίος
      (neo→next)←NULL  //τερματισμός λίστας
      current→next=neo  //σύνδεση στοιχείου
    ΤΕΛΟΣ
     ΑΛΛΙΩΣ ΑΡΧΗ  //ενδιάμεσος κόμβος
       (neo→next)←(current→next)
       (current→next)←neo //σύνδεση
     ΤΕΛΟΣ

current←start
    ΟΣΟ current<>NULL ΑΡΧΗ
         ΓΡΑΨΕ current→number
         current←(current→next)
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
     int number;
     struct komvos *next;
};
struct komvos *start, *current, *neo;

int main(void){
   //…….list created…….

     neo=(struct komvos*)malloc(sizeof(struct komvos));
     printf("give a number:");
      scanf("%d", &neo->number);fflush(stdin);

    current=start;
    while((current!=NULL)&&(current->number!=5))
          current=current->next;
    if(current->next==NULL){
          neo->next=NULL;
           current->next=neo;
         }
    else{
        neo->next=current->next;
        current->next=neo;
        }

  current=start;
  while(current!=NULL){
         printf("%d ",  current->number);
         current=current->next;
   }
getchar();
}
```

9.3.6.6 Διαγραφή πρώτου ή τελευταίου κόμβου

Πρόβλημα:

«Δίνεται συνδεδεμένη λίστα ακεραίων με δείκτη αρχής start. Να διαγραφεί ο πρώτος κόμβος της και ο τελευταίος κόμβος της»

Συζήτηση:

Εφόσον η λίστα δεν είναι κενή, η διαγραφή του πρώτου κόμβου της γίνεται με μια απλή μετακίνηση του δείκτη αρχής στον επόμενο κόμβο. Για να διαγραφεί ο τελευταίος κόμβος, θα πρέπει πρώτα να αναζητηθεί το τέλος της λίστας. Ωστόσο, δεν θα τοποθετηθεί ένας δείκτης ακριβώς στον τελευταίο κόμβο, αλλά στον προηγούμενό του, Η διαγραφή του τελευταίου κόμβου θα γίνει απλώς με την αποσύνδεσή του από τη λίστα. Ειδική περίπτωση είναι η περίπτωση να υπάρχει μόνο ένας κόμβος στη λίστα, οπότε δεν μπορεί να υπάρχει «προηγούμενος» κόμβος.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΓΡΑΦΗ_ΑΡΧΗ_ΤΕΛΟΣ
ΤΥΠΟΙ
    ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
```

```
            number:ΑΚΕΡΑΙΕΣ
            next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
    ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
    start, current, previous, neo: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
ΑΡΧΗ
// Έστω ότι έχει δημιουργηθεί μία συνδεδεμένη λίστα.
// Έστω ότι ο δείκτης start «δείχνει» στον πρώτο κόμβο της
// λίστας και ο δείκτης του τελευταίου κόμβου δείχνει σε NULL
    ΑΝ start=NULL ΤΟΤΕ
      ΓΡΑΨΕ 'ΔΙΑΓΡΑΦΗ ΑΔΥΝΑΤΗ'
    ΑΛΛΙΩΣ ΑΡΧΗ //μη κενή λίστα
      //διαγραφή 1ου κόμβου
      start←(start→next) //αποσύνδεση 1ου κόμβου
      current←start
      ΟΣΟ current<>NULL ΑΡΧΗ
          ΓΡΑΨΕ current→number
          current←(current→next)
      ΤΕΛΟΣ //της ΟΣΟ
    //διαγραφή τελευταίου κόμβου
    ΑΝ start<>NULL ΤΟΤΕ
     ΑΝ start→next=NULL ΤΟΤΕ //μόνο ένας κόμβος στη λίστα
        start=NULL
     ΑΛΛΙΩΣ ΑΡΧΗ
        current=start
        previous=current
        ΟΣΟ (current→next<>NULL)  ΑΡΧΗ
            previous←current //θέσε στο τρέχων
            current←(current→next)  //μετακίνησε το τρέχων
        ΤΕΛΟΣ //της ΟΣΟ…,
        (previous→next)←NULL //αποσύνδεση τελευταίου
     ΤΕΛΟΣ //στο ΑΛΛΙΩΣ…
  ΤΕΛΟΣ //στη μη κενή λίστα
  current←start
    ΟΣΟ current<>NULL ΑΡΧΗ
          ΓΡΑΨΕ current→number
          current←(current→next)
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
      int number;
      struct komvos *next;
};
struct komvos *start, *current, *neo, *previous;

int main(void){
   //……list created…….


  if(start==NULL)
    printf("EMPTY LIST");
  else{
    start=start->next;
    current=start;
    while(current!=NULL){
         printf("%d ",  current->number);
         current=current->next;
     }  printf("\n");


   if(start!=NULL)
     if(start->next==NULL)
       start=NULL;
     else{
        current=start;
        previous=current;
        while(current->next!=NULL) {
          previous=current;
          current=current->next;
        }
        previous->next=NULL;
      }
  }
  current=start;
  while(current!=NULL){
         printf("%d ",  current->number);
         current=current->next;
   }
getchar();
}
```

9.3.6.7 Διαγραφή κόμβου με συγκεκριμένη τιμή

Πρόβλημα:

«Δίνεται συνδεδεμένη λίστα ακεραίων με δείκτη αρχής start. Είναι γνωστό ότι κάποιος από τους κόμβους περιέχει το 5. Να διαγραφεί αυτός ο κόμβος»

Συζήτηση:

Σε αυτή την περίπτωση, γνωρίζουμε ότι η λίστα είναι μη κενή, ωστόσο δεν έχουμε ένδειξη για το εάν ο κόμβος που περιέχει το 5 είναι πρώτος, τελευταίος ή στη μέση. Επιπλέον, στην περίπτωση που υπάρχει μόνο αυτός ο κόμβος στη λίστα, η λίστα θα πρέπει να γίνει κενή. Και πάλι, είναι απαραίτητη η χρήση ενός επιπλέον δείκτη ο οποίος θα πρέπει να τοποθετηθεί στον προηγούμενο κόμβο από αυτόν που θα διαγραφεί.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΓΡΑΦΗ_ΚΟΜΒΟΥ
ΤΥΠΟΙ
```

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

```
    ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
            number:ΑΚΕΡΑΙΕΣ
            next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
    ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
    start, current, neo, previous: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
ΑΡΧΗ
// Έστω ότι έχει δημιουργηθεί μία συνδεδεμένη λίστα με μη
// μηδενικό πλήθος κόμβων. Έστω ότι ο δείκτης start «δείχνει»
// στον πρώτο κόμβο της λίστας και ο δείκτης του τελευταίου
// κόμβου δείχνει σε NULL
    ΑΝ start→next = NULL ΤΟΤΕ
      start←NULL //μόνο ένας κόμβος, η λίστα γίνεται κενή
    ΑΛΛΙΩΣ ΑΡΧΗ //η λίστα περιέχει >1 κόμβους
      ΑΝ start→number=5 ΤΟΤΕ //διαγραφή πρώτου κόμβου
          start←(start→next)
      ΑΛΛΙΩΣ ΑΡΧΗ
        current←start
        previous←current
       ΟΣΟ (current→next<>NULL)and(current→number<>5) ΑΡΧΗ
            previous←current
            current←(current→next)
       ΤΕΛΟΣ //της ΟΣΟ
      (previous→next)←(current→next) //αποσύνδεση
    ΤΕΛΟΣ
  ΤΕΛΟΣ   //λίστα με >1 κόμβους
 current←start
    ΟΣΟ current<>NULL ΑΡΧΗ
         ΓΡΑΨΕ current→number
         current←(current→next)
    ΤΕΛΟΣ
ΤΕΛΟΣ
```

```
typedef struct komvos{
     int number;
     struct komvos *next;
};
struct komvos *start, *current, *neo, *previous;

int main(void){
   //…….list created…….


  if(start->next==NULL)
    start=NULL;
  else{
    if(start->number==5)
      start=start->next;
    else{
      current=start;
      previous=current;
      while((current->next!=NULL)&&(current->number!=5)){
        previous=current;
         current=current->next;
       }
       previous->next=current->next;
     }
  }
  current=start;
  while(current!=NULL){
        printf("%d ",  current->number);
        current=current->next;
   }
getchar();
}
```

9.3.6.8 Δημιουργία δύο λιστών από μία (διαχωρισμός)

Πρόβλημα:

«Δίνεται συνδεδεμένη λίστα ακεραίων με δείκτη αρχής start. Να δημιουργηθούν δύο λίστες, έτσι ώστε η μία να περιέχει μόνο τους άρτιους και η δεύτερη μόνο τους περιττους αριθμούς της αρχικής λίστας»

Συζήτηση:

Η εφαρμογή δεν παρουσιάζει ιδιαίτερες δυσκολίες. Θα διαπεράσουμε την αρχική λίστα και θα προσθέτουμε έναν νέο κόμβο με την κατάλληλη τιμή σε μία από τις δύο νέες λίστες, ανάλογα με την τιμή του τρέχοντος κόμβου. Προφανώς θα απαιτηθούν τρεις τρέχοντες δείκτες, ένας για κάθε λίστα. Επίσης, θα απαιτηθούν δύο νέοι δείκτες αρχής για τις δύο νέες λίστες.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΧΩΡΙΣΜΟΣ
ΤΥΠΟΙ
    ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
            number:ΑΚΕΡΑΙΕΣ
            next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
    ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
    start, current, neo, L1, L2, c1, c2: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
ΑΡΧΗ
// Έστω ότι έχει δημιουργηθεί μία συνδεδεμένη λίστα με μη
// μηδενικό πλήθος κόμβων. Έστω ότι ο δείκτης start «δείχνει»
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
     int number;
     struct komvos *next;
};
struct komvos *start, *current, *neo;
 struct komvos  *previous,*l1,*l2,*c1,*c2;

int main(void){
   //…….list created…….
```

```
// στον πρώτο κόμβο της λίστας και ο δείκτης του τελευταίου
// κόμβου δείχνει σε NULL
   ΑΝ start = NULL ΤΟΤΕ
     ΓΡΑΨΕ 'ΛΙΣΤΑ ΚΕΝΗ'
   ΑΛΛΙΩΣ ΑΡΧΗ //η λίστα περιέχει >1 κόμβους
     L1 ←NULL
     L2← NULL
     C1 ←NULL
     C2←NULL
     current←start
     ΟΣΟ current<>NULL ΑΡΧΗ
        neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
        (neo→next) ←NULL
        (neo→number)←(current→number)

        ΑΝ current→number mod 2=0 ΤΟΤΕ
           ΑΝ L1=NULL ΤΟΤΕ ΑΡΧΗ
              L1←neo
              c1←L1
           ΤΕΛΟΣ
           ΑΛΛΙΩΣ ΑΡΧΗ
              (c1→next)=neo
              c1←(c1→next)
           ΤΕΛΟΣ
        ΑΛΛΙΩΣ
           ΑΝ L2=NULL ΤΟΤΕ ΑΡΧΗ
              L2←neo
              c2←L2
           ΤΕΛΟΣ
           ΑΛΛΙΩΣ ΑΡΧΗ
              (c2→next)=neo
              c2←(c2→next)
           ΤΕΛΟΣ
        current←(current→next) //επόμενο στοιχείο
     ΤΕΛΟΣ //ΟΣΟ…
  ΤΕΛΟΣ //μη κενή λίστα
//εκτύπωση των τριών λιστών…..
ΤΕΛΟΣ
```

```
 if(start==NULL)
    printf("LISTEMPTY");
 else{
   l1=NULL;l2=NULL;c1=NULL;c2=NULL;


    current=start;
    while(current!=NULL){
       neo=(struct komvos*)malloc(sizeof(struct kom-
vos));
       neo->next=NULL;
       neo->number=current->number;

       if(current->number%2==0)
        if(l1==NULL){
          l1=neo;
          c1=neo;
         }
         else{
          c1->next=neo;
          c1=c1->next;
          }
       else
        if(l2==NULL){
          l2=neo;
          c2=neo;
        }
         else{
          c2->next=neo;
          c2=c2->next;
          }
   current=current->next;
   }
}
  //print three lists
getchar();
}
```

9.3.6.9 Συνένωση λιστών

Πρόβλημα:

«Δίνονται δύο λίστες ακεραίων με δείκτες αρχής start1 και start2. Να συνενωθούν οι λίστες έτσι ώστε όλα τα στοιχεία της 2^{ης} να έπονται των στοιχείων της πρώτης»

Συζήτηση:

Θα διαπεράσουμε την πρώτη λίστα, θέτοντας ένα δείκτη στον τελευταίο κόμβο της και στη συνέχεια θα θέσουμε το δείκτη αυτού του κόμβου να δείχνει εκεί όπου δείχνει ο δείκτης αρχής της δεύτερης λίστας. Θα πρέπει να ελεγχθούν οι περιπτώσεις των κενών λιστών (ένα μία λίστα είναι κενή, τότε η συνενωμένη λίστα αποτελείται μόνο από την άλλη λίστα). Θα χρησιμοποιηθεί ο δείκτης start1 και ως δείκτης αρχής της τελικής, συνενωμένης λίστας.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΣΥΝΕΝΩΣΗ
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
```

```
            number:ΑΚΕΡΑΙΕΣ
            next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
    ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
    start1, current, neo, start2: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
ΑΡΧΗ
// Έστω ότι έχουν δημιουργηθεί δύο λίστες με δείκτες αρχής
// start1 και start2 αντίστοιχα. Οι λίστες εκτυπώνονται…

   ΑΝ (start1 = NULL and start2=NULL) ΤΟΤΕ
     ΓΡΑΨΕ 'ΛΙΣΤΕΣ ΚΕΝΕΣ'
   ΑΛΛΙΩΣ
     ΑΝ start1=NULL ΤΟΤΕ //η 1η είναι κενή
          start1=start2
     ΑΛΛΙΩΣ  ΑΡΧΗ
       //Και οι δύο λίστες έχουν δεδομένα
       // Αν η 2η είναι κενή, τότε η συνένωση είναι ίδια με την
//πρώτη λίστα.
            current=start1
            ΟΣΟ current→next<>NULL
               current←(current→next) //τέλος 1ης

            (current→next)←start2 //σύνδεση λιστών
      ΤΕΛΟΣ
//τελική εκτύπωση λίστας με δείκτη αρχής start1
ΤΕΛΟΣ
```

```
      int number;
      struct komvos *next;
};
struct komvos *start1, *start2, *current, *neo;

int main(void){
    //…… 2 lists created……
   //……..2 lists printed……

  if((start1==NULL)&&(start2==NULL))
     printf("LISTS EMPTY");
  else
    if(start1==NULL)
      start1=start2;
    else{
       current=start1;
       while(current->next!=NULL)
          current=current->next;

       current->next=start2;
      }


//print list from start1
getchar();
}
```

9.3.6.10 Απόφαση σε συνδεδεμένη λίστα

Πρόβλημα:

«Δίνεται συνδεδεμένη λίστα ακεραίων με δείκτη αρχής start. Να βρεθεί εάν οι κόμβοι της περιέχουν αριθμούς ταξινομημένους κατά αύξουσα σειρά»

Συζήτηση:

Η επίλυση του προβλήματος είναι παρόμοια με αυτή των πινάκων. Θα χρησιμοποιηθεί μια λογική μεταβλητή, με την οποία θα ελέγχουμε εάν οι γειτονικοί κόμβοι είναι στη σωστή σειρά. Αρκεί μία διαπέραση της λίστας.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΑΠΟΦΑΣΗ
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
            number:ΑΚΕΡΑΙΕΣ
            next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   start, current: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   ταξ:ΛΟΓΙΚΕΣ
ΑΡΧΗ
// Έστω ότι έχει δημιουργηθεί μία συνδεδεμένη λίστα.
// Έστω ότι ο δείκτης start «δείχνει» στον πρώτο κόμβο της
   ΑΝ start=NULL ΤΟΤΕ
     ΓΡΑΨΕ 'ΑΔΥΝΑΤΟ'
   ΑΛΛΙΩΣ ΑΡΧΗ //μη κενή λίστα
     ταξ←true
     current←start
     ΟΣΟ (current→next<>NULL and ταξ=true) ΑΡΧΗ
       ΑΝ (current→number)>(current→next→number) ΤΟΤΕ
            ταξ←false
       current←(current→next)
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
      int number;
      struct komvos *next;
};
struct komvos *start, *current, *neo, *previous;
int t;
int main(void){
   //…….list created…….

  if(start==NULL)
    printf("EMPTY LIST");
  else{
    t=1;
    current=start;
    while((current->next!=NULL)&&(t==1)){
         if(current->number>current->next->number)
              t=0;
         current=current->next;
     }
```

ΤΕΛΟΣ *//της ΟΣΟ* **ΑΝ** ταξ=**true ΤΟΤΕ ΑΡΧΗ** **ΓΡΑΨΕ** 'ΛΙΣΤΑ ΤΑΞΙΝΟΜΗΜΕΝΗ' current←start **ΟΣΟ** current<>**NULL ΑΡΧΗ** **ΓΡΑΨΕ** current→number current←(current→next) **ΤΕΛΟΣ** **ΤΕΛΟΣ** **ΑΛΛΙΩΣ** **ΓΡΑΨΕ** ΄ΛΙΣΤΑ ΜΗ ΤΑΞΙΝΟΜΗΜΕΝΗ' **ΤΕΛΟΣ** **ΤΕΛΟΣ** **ΤΕΛΟΣ**	if(t){ printf("list sorted\n"); current=start; while(current!=NULL) { printf("%d ",current->number); current=current->next; } } else printf("list not sorted"); } getchar(); }

9.3.6.11 Μια εφαρμογή (διαχείριση συνόλων)

Πρόβλημα:

«Δίνεται συνδεδεμένη λίστα ακεραίων L με δείκτη αρχής start. Να βρεθεί εάν η λίστα μπορεί να αναπαριστά ένα σύνολο. Εάν δοθεί ένας ακέραιος x, να βρεθεί εάν $x \in L$ *.Εάν δεν ανήκει, τότε να εισαχθεί στο σύνολο. Εάν δοθεί ακόμη μία λίστα Q με δείκτη αρχής start2, να βρεθούν τα* $L \cup Q, L \cap Q$ *(η ένωση και η τομή των L και Q)»*

Συζήτηση:

Η εφαρμογή αυτή είναι κάπως πιο περίπλοκη από αυτές με τις οποίες ασχοληθήκαμε έως τώρα. Ωστόσο, πρόκειται για μια ξεκάθαρη περίπτωση όπου δεν είναι δυνατόν να χρησιμοποιηθεί πίνακας. Τα σύνολα δεν έχουν κάποιο συγκεκριμένο μέγεθος, καθώς μπορούν να μεγαλώνουν και να μικραίνουν με την προσθήκη ή διαγραφή νέων στοιχείων αντίστοιχα. Όπως είναι φανερό, ο καλύτερος τρόπος για να αναπαραστήσουμε ένα σύνολο είναι με τη βοήθεια μιας συνδεδεμένης λίστας. Για την εφαρμογή αυτή, θα κατασκευάσουμε έναν αλγόριθμο για κάθε ένα διαφορετικό ερώτημα.

1^{ο} ερώτημα: Μία λίστα μπορεί να αναπαριστά ένα σύνολο όταν τα δεδομένα που είναι αποθηκευμένα στους κόμβους είναι όλα διαφορετικά μεταξύ τους. Άρα, το πρόβλημα είναι ένα πρόβλημα απόφασης, στο οποίο πρέπει να διατρέξουμε τη λίστα L και για κάθε κόμβο της να ελέγξουμε εάν τα δεδομένα του υπάρχουν ήδη στη λίστα. Σε μια τέτοια περίπτωση, η λίστα δεν μπορεί να είναι σύνολο.

Για το σκοπό αυτό, χρειαζόμαστε έναν τρέχοντα δείκτη ο οποίος θα χρησιμοποιείται κάθε φορά για να δείχνει στον κόμβο τον οποίο επισκεπτόμαστε. Ο δείκτης αυτός θα ξεκινήσει από τον 2^{ο} κόμβο, καθώς εάν υπάρχει μόνο ένας κόμβος στη λίστα, τότε αυτή είναι οπωσδήποτε σύνολο. Επιπλέον, θα χρησιμοποιηθεί ακόμη ένας δείκτης, ο οποίος θα ξεκινά από την αρχή της λίστας και θα καταλήγει στον τρέχοντα δείκτη, έτσι ώστε να ελέγχεται η «έως τώρα» ισότητα των δεδομένων των κόμβων.

Λύση:

ΑΛΓΟΡΙΘΜΟΣ ΣΥΝΟΛΟ **ΤΥΠΟΙ** **ΕΓΓΡΑΦΗ** ΚΟΜΒΟΣ **ΑΡΧΗ** number:**ΑΚΕΡΑΙΕΣ** next: **ΔΕΙΚΤΗΣ_ΣΕ** ΚΟΜΒΟΣ	#include <stdio.h> #include <stdlib.h> typedef struct komvos{ int number; struct komvos *next;

```
    ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
    start, current,c1: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
    set:ΛΟΓΙΚΕΣ
ΑΡΧΗ
// Έστω ότι έχει δημιουργηθεί μία συνδεδεμένη λίστα.
// Έστω ότι ο δείκτης start «δείχνει» στον πρώτο κόμβο της
   ΑΝ start=NULL ΤΟΤΕ
     ΓΡΑΨΕ 'ΚΕΝΟ ΣΥΝΟΛΟ'
   ΑΛΛΙΩΣ
     ΑΝ start->next=NULL ΤΟΤΕ
       ΓΡΑΨΕ 'ΣΥΝΟΛΟ ΜΕ ΕΝΑ ΣΤΟΙΧΕΙΟ'
     ΑΛΛΙΩΣ ΑΡΧΗ
       current←(start→next)
       set←true
       ΟΣΟ current<>NULL and set=true ΑΡΧΗ
         c1=start;
         ΟΣΟ c1<>current ΑΡΧΗ
           ΑΝ c1->number=current->number ΤΟΤΕ
             set←false
           c1←(c1→next)
         ΤΕΛΟΣ //ΟΣΟ…
         current←(current→next)
       ΤΕΛΟΣ //ΟΣΟ..
       ΑΝ set=true ΤΟΤΕ
         ΓΡΑΨΕ 'ΕΙΝΑΙ ΣΥΝΟΛΟ'
       ΑΛΛΙΩΣ
         ΓΡΑΨΕ 'ΔΕΝ ΕΙΝΑΙ ΣΥΝΟΛΟ'
     ΤΕΛΟΣ //ΑΛΛΙΩΣ…
ΤΕΛΟΣ
```

```
};
struct komvos *start, *current, *c1, *neo;
int set;
int main(void){
   //……list created…….


 if(start==NULL)
  printf("EMPTY SET");
 else
  if(start->next==NULL)
   printf("SET with ONE ELEMENT");
  else{
   current=start->next;
   set=1;
   while((current!=NULL)&&(set)){
     c1=start;
     while(c1!=current){
       if(c1->number==current->number)
         set=0;
       c1=c1->next;
       }
      current=current->next;
    }
    if(!set)
      printf("NOT A SET\n");
    else
      printf("SET!!\n");
  }
getchar();
}
```

2ο ερώτημα: Εφόσον έχουμε αποφασίσει ότι η λίστα μας αναπαριστά ένα σύνολο, εάν δοθεί ένα νέο στοιχείο σε έναν κόμβο, τότε μπορούμε να εισάγουμε αυτό το στοιχείο στο σύνολο, συνδέοντάς το στη συνδεδεμένη λίστα. Είναι προφανές ότι και πάλι θα πρέπει να διατρέξουμε τη λίστα ώστε να βεβαιωθούμε οτι το στοιχείο δεν υπάρχει ήδη μέσα στο σύνολο. Η προσθήκη του νέου κόμβου μπορεί να γίνει είτε στην αρχή είτε στο τέλος της λίστας, καθώς στα σύνολα δεν υφίσταται σειρά στα στοιχεία.

```
ΑΛΓΟΡΙΘΜΟΣ ΕΙΣΑΓΩΓΗ_ΣΥΝΟΛΟ
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
           number:ΑΚΕΡΑΙΕΣ
           next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
   start, current,c1, neo: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   set, exists:ΛΟΓΙΚΕΣ
ΑΡΧΗ
// Έστω ότι έχει δημιουργηθεί μία συνδεδεμένη λίστα.
// Έστω ότι ο δείκτης start «δείχνει» στον πρώτο κόμβο της
//Έστω ότι η λίστα αποτελεί σύνολο (έχει ήδη ελεγχθεί)

     neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
     ΔΙΑΒΑΣΕ neo→number
     exists←false
     current←start
     ΟΣΟ (current<>NULL) ΑΡΧΗ
       ΑΝ current->number=neo->number ΤΟΤΕ
         exists←true
       current←(current→next)
     ΤΕΛΟΣ
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
     int number;
     struct komvos *next;
};
struct komvos *start, *current, *c1, *neo;
int set, exists;
int main(void){
   //……list created…….
   //……it is a set……

 neo=(struct komvos*)malloc(sizeof(struct komvos));
 printf("give element:");scanf("%d",&neo->number);
 fflush(stdin);
 exists=0;
 current=start;
 while(current!=NULL){
    if(current->number==neo->number)
      exists=1;
    current=current->next;
 }
```

```
      ΑΝ (not exists) ΤΟΤΕ ΑΡΧΗ
           neo->next=start;
           start=neo; //σύνδεση στην αρχή
       ΤΕΛΟΣ
      ΑΛΛΙΩΣ
          ΓΡΑΨΕ 'ΤΟ ΣΤΟΙΧΕΙΟ ΥΠΑΡΧΕΙ ΗΔΗ'
ΤΕΛΟΣ
```

```
  if(!exists){
     neo->next=start;
     start=neo;
  }
  else
      printf("element already exists\n");
getchar();
}
```

3ο ερώτημα (η ένωση δύο συνόλων): Η περίπτωση αυτή είναι κάπως πιο περίπλοκη. Εφόσον έχουμε εισάγει δύο λίστες και έχουμε αποφασίσει οτι και οι δύο είναι σύνολα, τότε η ένωση των δύο συνόλων προκύπτει από τη συνένωση των δύο λιστών. Ωστόσο, η χρήση του ήδη έτοιμου αλγόριθμου δεν επαρκεί, καθώς θα πρέπει και η νέα λίστα να μη συμπεριλαμβάνει όμοια στοιχεία. Στη νέα, συνενωμένη λίστα, θα πρέπει ξεκινώντας από το δεύτερο σύνολο και επισκεπτόμενοι κάθε κόμβο, να ελέγξουμε όλα τα στοιχεία της πρώτης λίστας για πιθανή ισότητα. Σε μια τέτοια περίπτωση θα πρέπει να παρακάμπτεται ο ένας από τους κόμβους με το ίδιο στοιχείο. Η λύση αυτή βέβαια είναι αρκετά περίπλοκη!
Εναλλακτικά, θα μπορούσαμε να εφαρμόσουμε τον αλγόριθμο εισαγωγής στοιχείου σε σύνολο, διατρέχοντας τη δεύτερη λίστα αντί να διαβάζουμε το νέο στοιχείο. Η λύση αυτή είναι αρκετά απλούστερη και αυτή θα εφαρμόσουμε στη συνέχεια.

```
ΑΛΓΟΡΙΘΜΟΣ ΕΝΩΣΗ_ΣΥΝΟΛΩΝ
ΤΥΠΟΙ
    ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
             number:ΑΚΕΡΑΙΕΣ
             next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
    ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
  start,start2,current,current2, neo: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
  set, exists:ΛΟΓΙΚΕΣ
ΑΡΧΗ
// Έστω ότι έχουμε δημιουργήσει δύο σύνολα-λίστες.
//Έστω οτι ο start δείχνει στην 1η λίστα και ο
//start2 δείχνει στη 2η λίστα.

   current2=start2
   ΟΣΟ current2<>NULL ΑΡΧΗ
     neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
     (neo→number)←(current2→number)
     exists←false
     current←start
     ΟΣΟ (current<>NULL) ΑΡΧΗ
        ΑΝ current->number=neo->number ΤΟΤΕ
           exists←true
        current←(current→next)
     ΤΕΛΟΣ
     ΑΝ (not exists) ΤΟΤΕ ΑΡΧΗ
          neo->next=start;
          start=neo; //σύνδεση στην αρχή
     ΤΕΛΟΣ
     current2←(current2→next)
   ΤΕΛΟΣ //εξωτερικής ΟΣΟ…
//τύπωσε λίστα start
ΤΕΛΟΣ
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
     int number;
     struct komvos *next;
};
struct komvos *start, *start2,*current, *current2, *neo;
int set, exists;
int main(void){
   //……2 lists created……..
   //……they are sets……



current2=start2;
while(current2!=NULL){
 neo=(struct komvos*)malloc(sizeof(struct komvos));
 neo->number=current2->number;
 exists=0;
 current=start;
 while(current!=NULL){
    if(current->number==neo->number)
       exists=1;
    current=current->next;
 }
 if(!exists){
    neo->next=start;
    start=neo;
 }
 current2=current2->next;
}
//print start list
getchar();
}
```

4ο ερώτημα (η τομή δύο συνόλων):Η τομή δύο συνόλων είναι ένα νέο σύνολο που περιέχει μόνο τα στοιχεία που είναι κοινά στα δύο σύνολα. Θα δημιουργήσουμε

λοιπόν μία νέα λίστα, στην οποία θα προσθέτουμε κόμβους για τους οποίους έχουμε ελέγξει οτι περιέχουν τον ίδιο αριθμό. Θα διατρέξουμε την $1^η$ λίστα και για κάθε στοιχείο της θα ελέγχουμε όλη τη $2^η$ λίστα. Εάν υπάρχει και εκεί, τότε θα προσθέτουμε αυτό το στοιχείο στην τομή. Σημειώνεται οτι με τον τρόπο αυτό δεν απαιτείται έλεγχος ίδιων στοιχείων στο νέο σύνολο (την τομή).

```
ΑΛΓΟΡΙΘΜΟΣ ΤΟΜΗ_ΣΥΝΟΛΩΝ
ΤΥΠΟΙ
    ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
            number:ΑΚΕΡΑΙΕΣ
            next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
    ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
  start,start2,current,current2, neo: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
  start3: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
ΑΡΧΗ
// Έστω ότι έχουμε δημιουργήσει δύο σύνολα-λίστες.
//Έστω οτι ο start δείχνει στην 1η λίστα και ο
//start2 δείχνει στη 2η λίστα.
   start3←NULL;
   current←start;
   ΟΣΟ current<>NULL ΑΡΧΗ
      current2←start2;
      ΟΣΟ current2<>NULL ΑΡΧΗ
         ΑΝ current→number=current2→number ΤΟΤΕ ΑΡΧΗ
            neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
            (neo→number)←(current2→number)
            (neo→next)←start3
            start3=neo
         ΤΕΛΟΣ
         current2←(current2→next)
      ΤΕΛΟΣ //ΟΣΟ…
      current←(current→next)
   ΤΕΛΟΣ //εξωτερικής ΟΣΟ…

   ΑΝ start3<>NULL ΤΟΤΕ ΑΡΧΗ
       //τύπωσε λίστα start3
   ΑΛΛΙΩΣ
      ΓΡΑΨΕ 'Η ΤΟΜΗ ΕΙΝΑΙ ΤΟ ΚΕΝΟ ΣΥΝΟΛΟ'
ΤΕΛΟΣ
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
     int number;
     struct komvos *next;
};
struct komvos *start, *start2,*current, *current2,
*neo,*start3;

int main(void){
   //……2 lists created……..
   //……they are sets……

start3=NULL;
current=start;
while(current!=NULL){
  current2=start2;
  while(current2!=NULL){
    if(current->number==current2->number){
      neo=(struct komvos*)malloc(sizeof(struct komvos));
      neo->number=current2->number;
      neo->next=start3;
      start3=neo;
     }
    current2=current2->next;
   }
   current=current->next;
 }

 if(start3!=NULL)
      //print start3 list
 else
     printf("Intersection is the EMPTY SET");
getchar();
}
```

9.3.6.12 Συμπίεση αραιού πίνακα

Πρόβλημα:

«Ένας πίνακας λέγεται αραιός εάν περιέχει μηδενικά σε ποσοστό μεγαλύτερο του 80% του συνολικού πλήθους των στοιχείων του. Να εισαχθεί ένας δισδιάστατος πίνακας A_{10X50} και εφόσον είναι αραιός να συμπιεστεί κατάλληλα σε μία συνδεδεμένη λίστα»

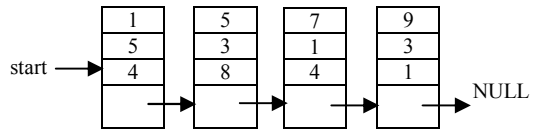
Συζήτηση:

Μετά την εισαγωγή του πίνακα, ο έλεγχος για το εάν είναι αραιός ή όχι είναι απλή υπόθεση: θα μετρήσουμε τα μη μηδενικά στοιχεία και θα αποφασίσουμε εάν ξεπερνούν το 20% του συνόλου των στοιχείων. Ο έλεγχος αυτός είναι πολύ απλός και δεν θα ασχοληθούμε περισσότερο.

Η συμπίεση σε συνδεδεμένη λίστα μπορεί να γίνει αποθηκεύοντας μόνο τα μη μηδενικά στοιχεία. Ωστόσο, στον πίνακα κάθε στοιχείο χαρακτηρίζεται από τη θέση του που περιλαμβάνει τον αριθμό γραμμής και τον αριθμό

στήλης στις οποίες βρίσκεται. Κατά συνέπεια, εκτός από την τιμή του, για κάθε στοιχείο θα αποθηκεύσουμε και αυτές τις δύο τιμές της θέσης του. Έτσι, ο βασικός κόμβος της συνδεδεμένης λίστας θα αποτελείται συνολικά από τέσσερα πεδία (τρεις ακεραίους και τον δείκτη στον επόμενο κόμβο).
Σχηματικά, αν υποθέσουμε ότι ο πίνακας μας έχει 10 γραμμές και 5 στήλες και μόνο 4 μη μηδενικά στοιχεία, η λίστα που τον αναπαριστά φαίνεται παρακάτω:

	1	2	3	4	5
1	0	0	0	0	4
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	8	0	0
6	0	0	0	0	0
7	4	0	0	0	0
8	0	0	0	0	0
9	0	0	1	0	0
10	0	0	0	0	0



Όπως είναι φανερό, διατρέξαμε γραμμή προς γραμμή τον πίνακα και όπου βρέθηκε μη μηδενικό στοιχείο αποθηκεύτηκε μέσα στη λίστα.

Μετά τη δημιουργία της λίστας, η αποσυμπίεση προκύπτει με μια απλή διαπέραση της λίστας και ενημέρωση ενός πίνακα ο οποίος έχει αρχικοποιηθεί με μηδενικά σε όλα τα στοιχεία του.

Λύση:

```
ΑΛΓΟΡΙΘΜΟΣ ΣΥΜΠΙΕΣΗ_ΠΙΝΑΚΑ
ΤΥΠΟΙ
   ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
           grammi:ΑΚΕΡΑΙΕΣ
           stili:ΑΚΕΡΑΙΕΣ
           number:ΑΚΕΡΑΙΕΣ
           next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
   ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
  start,current, neo: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
  A[10,50], i, j:ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
  Ιιεισαγωγή πίνακα Α, έλεγχος για αραιό
  start←NULL
  current←start
  ΓΙΑ i←1 ΜΕΧΡΙ 10
     ΓΙΑ j←1 ΜΕΧΡΙ 50
          ΑΝ A[i,j]<>0 ΤΟΤΕ ΑΡΧΗ
             neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
             (neo→number)←A[i,j]
             (neo→grammi)←i
             (neo→stili)←j
             (neo→next)←NULL
             ΑΝ start=NULL ΤΟΤΕ ΑΡΧΗ
                start←neo
                current←start
             ΤΕΛΟΣ
             ΑΛΛΙΩΣ ΑΡΧΗ
                (current→next)←neo
                current←neo
             ΤΕΛΟΣ
          ΤΕΛΟΣ
//ο πίνακας συμπιέστηκε. Εκτύπωση λίστας
// ακολουθεί η αποσυμπίεση. Μηδενίζουμε τον Α και
//τον ξαναδημιουργούμε
```

```
#include <stdio.h>
#include <stdlib.h>
typedef struct komvos{
     int grammi;
     int stili;
     int number;
     struct komvos *next;
};
struct komvos *start, *current, *neo;
int a[11][6], i, j; //smaller array
int main(void){
// array a[][] has been read and is sparse
   start=NULL;
   current=start;
   for(i=1;i<=10;i++)
      for(j=1;j<=5;j++)
         if(a[i][j]!=0){
            neo=(struct komvos*)malloc(sizeof(struct komvos));
            neo->number=a[i][j];
            neo->grammi=i;
            neo->stili=j;
            neo->next=NULL;
            if(start==NULL){
               start=neo;
               current=start;
               }
            else{
               current->next=neo;
               current=neo;
               }
         }

//print list from start

  for(i=1;i<=10;i++)
```

ΓΙΑ i←1 **ΜΕΧΡΙ** 10 **ΓΙΑ** j←1 **ΜΕΧΡΙ** 50 A[i,j]←0 current=start **ΟΣΟ** current<>**NULL ΑΡΧΗ** A[current→grammi,current→stili]←(current→number) current←(current→next) ΤΕΛΟΣ //*τύπωσε πίνακα Α αποσυμπιεσμένο* **ΤΕΛΟΣ**	for(j=1;j<=5;j++) a[i][j]=0; current=start; while(current!=NULL){ a[current->grammi][current->stili]=current->number; current=current->next; } *//print array again* getchar(); }

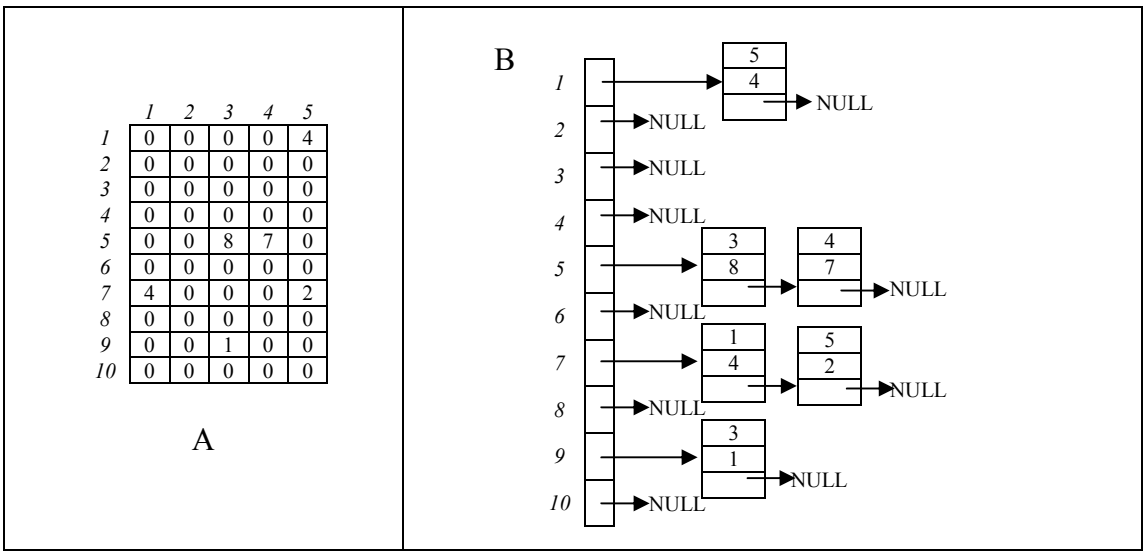
9.3.6.13 Συμπίεση πίνακα – μικτή δομή

Πρόβλημα:

«Δίνεται αραιός δισδιάστατος πίνακας ακεραίων A_{MXN}. Να συμπιεστεί ο πίνακας, χρησιμοποιώντας έναν μονοδιάστατο πίνακα B_M, κάθε στοιχείο του οποίου είναι μια συνδεδεμένη λίστα που αντιστοιχεί στα μη μηδενικά στοιχεία της i-στής γραμμής του Α (i=1, 2, ...M)»

Συζήτηση:

Μολονότι η εκφώνηση φαίνεται κάπως περίπλοκη, εάν περιγράψουμε σχηματικά το στόχο μας, θα γίνει πολύ σαφέστερη. Για έναν δισδιάστατο πίνακα με Μ γραμμές θα κατασκευάσουμε έναν μονοδιάστατο με Μ στοιχεία. Κάθε στοιχείο του μονοδιάστατου θα είναι ο δείκτης αρχής μιας συνδεδεμένης λίστας. Κάθε κόμβος της λίστας θα αποθηκεύει την τιμή του στοιχείου και τη στήλη στην οποία αυτό βρίσκεται:



Παρά το γεγονός ότι η συμπίεση δεν είναι τόσο «φανερή», θα πρέπει να θυμόμαστε οτι ο τρόπος αυτός αφορά σε μεγάλους πίνακες όπου το πλήθος των μη μηδενικών στοιχείων είναι μικρό!

Λύση:

ΑΛΓΟΡΙΘΜΟΣ ΣΥΜΠΙΕΣΗ_ΠΙΝΑΚΑ	#include <stdio.h>

```
ΤΥΠΟΙ
    ΕΓΓΡΑΦΗ ΚΟΜΒΟΣ ΑΡΧΗ
            stili:ΑΚΕΡΑΙΕΣ
            number:ΑΚΕΡΑΙΕΣ
            next: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
    ΤΕΛΟΣ
ΜΕΤΑΒΛΗΤΕΣ
  current, neo, B[10]: ΔΕΙΚΤΗΣ_ΣΕ ΚΟΜΒΟΣ
  A[10,5] i, j:ΑΚΕΡΑΙΕΣ
ΑΡΧΗ
  //εισαγωγή πίνακα Α
  ΓΙΑ i←1 ΜΕΧΡΙ 10
     B[i]←NULL //αρχικοποίηση πίνακα Β

  ΓΙΑ i←1 ΜΕΧΡΙ 10 ΑΡΧΗ
     current←B[i]
     ΓΙΑ j←1 ΜΕΧΡΙ 5
           ΑΝ A[i,j]<>0 ΤΟΤΕ ΑΡΧΗ
              neo←ΔΕΣΜΕΥΣΕ_ΧΩΡΟ(ΜΕΓΕΘΟΥΣ(ΚΟΜΒΟΣ))
              (neo→number)←A[i,j]
              (neo→stili)←j
              (neo→next)←NULL
              ΑΝ B[i]=NULL ΤΟΤΕ ΑΡΧΗ
                 B[i]←neo
                 current←B[i]
              ΤΕΛΟΣ
              ΑΛΛΙΩΣ ΑΡΧΗ
                 (current→next)←neo
                 current←neo
              ΤΕΛΟΣ
           ΤΕΛΟΣ
   ΤΕΛΟΣ
//ο πίνακας συμπιέστηκε. Εκτύπωση λίστας
  ΓΙΑ i←1 ΜΕΧΡΙ 10
    ΑΝ B[i]<>NULL ΤΟΤΕ ΑΡΧΗ
      current=B[i]
      ΟΣΟ current<>NULL ΑΡΧΗ
         ΓΡΑΨΕ current→stili, current→number
         current←(current→next)
      ΤΕΛΟΣ
    ΤΕΛΟΣ

// ακολουθεί η αποσυμπίεση. Μηδενίζουμε τον Α και
//τον ξαναδημιουργούμε
   ΓΙΑ i←1 ΜΕΧΡΙ 10
     ΓΙΑ j←1 ΜΕΧΡΙ 50
          A[i,j]←0

   ΓΙΑ i←1 ΜΕΧΡΙ 10
     ΑΝ B[i]<> NULL ΤΟΤΕ ΑΡΧΗ
          current←B[i]
          ΟΣΟ current<>NULL ΑΡΧΗ
            A[i,current→stili]←(current→number)
            current←(current→next)
          ΤΕΛΟΣ
     ΤΕΛΟΣ
//τύπωσε πίνακα Α αποσυμπιεσμένο
ΤΕΛΟΣ
```

```
#include <stdlib.h>
typedef struct komvos{
     int grammi;
     int stili;
     int number;
     struct komvos *next;
};
struct komvos *start, *current, *neo, *b[11];
int a[11][6], i, j; //smaller array
int main(void){
// array a[][] has been read and is sparse
  for(i=1;i<=10;i++)
     b[i]=NULL;

  for(i=1;i<=10;i++){
    current=b[i];
    for(j=1;j<=5;j++)
       if(a[i][j]!=0){
          neo=(struct komvos*)malloc(sizeof(struct komvos));
          neo->number=a[i][j];
          neo->stili=j;
          neo->next=NULL;
          if(b[i]==NULL){
            b[i]=neo;
            current=b[i];
            }
          else{
            current->next=neo;
            current=neo;
            }
       }
  }
//print list from start
 for(i=1;i<=10;i++){
   if(b[i]!=NULL){
     current=b[i];
     while(current!=NULL){
       printf("gr:%d,st:%d,data:%d ",i,current->stili,current->number);
       current=current->next;
     }
   }printf("\n");
 }

for(i=1;i<=10;i++)
     for(j=1;j<=5;j++)
         a[i][j]=0;


for(i=1;i<=10;i++)
  if(b[i]!=NULL){
   current=b[i];
   while(current!=NULL){
    a[i][current->stili]=current->number;
    current=current->next;
    }
  }
//print array again
getchar();
}
```

Ξένη Βιβλιογραφία

Knuth D (1973). The art of computer programming, Part I, Fundumental algorithms, Addison Wesley

Knuth D (1973). The art of computer programming, Part II, Seminumerical Algorithms, Addison Wesley

Thomas Cormen, Charles Leiserson , Ronald Rivest, Clifford Stein (2003). Introduction to Algorithms, McGraw-Hill Science/Engineering/Math

Steven S. Skiena, (2008). The Algorithm Design Manual, Springer

Ελληνική Βιβλιογραφία

Βακάλη, Γιαννόπουλος, κλπ. (1999). Ανάπτυξη Εφαρμογών σε προγραμματιστικό περιβάλλον, Παιδαγωγικό Ινστιτούτο.

Καμέας Αχιλλέας (2000), Τεχνικές Προγραμματισμού, Ελληνικό Ανοικτό Πανεπιστήμιο.

Μανωλόπουλος Ιωάννης, Δομές Δεδομένων. (Τόμος Α΄), Art of Text

Μανωλόπουλος Ιωάννης, Δομές Δεδομένων. (Τόμος Β΄), Art of Text

Χατζηλυγερούδης Ι. (1999). Δομές Δεδομένων, Ελληνικό Ανοικτό Πανεπιστήμιο

Λουκάκης, Μ. (1998). Δομές Δεδομένων, Αλγόριθμοι, Εκδ. Ζυγός