Δομές Δεδομένων

4η Διάλεξη

Στοιχειώδεις Δομές Δεδομένων: Πίνακες και Λίστες

Ε. Μαρκάκης

Εργαστήρια

- Ώρες εργαστηρίων
  - Τέσσερα τμήματα εργαστηρίων
    - XXXX001-XXXX060, Δευτέρα 09:00-11:00 (CSLAB II)
    - XXXX061-XXXX120, Δευτέρα 11:00-13:00 (CSLAB II)
    - XXXX121-XXXX180, Πέμπτη 11:00-13:00 (CSLAB II)
    - XXXX181-XXXX999, Πέμπτη 15:00-17:00 (CSLAB II)
  - Τα εργαστήρια είναι προαιρετικά
  - Σύντομες προγραμματιστικές ασκήσεις
  - Υλοποίηση βασικών δομών δεδομένων
  - Επίλυση αποριών και παροχή βοήθειας για τις εργασίες
  - Θέμα 1^{ου} εργαστηρίου: Υλοποίηση Αφηρημένου Τύπου Δεδομένων (ΑΤΔ) για αναπαράσταση μιγαδικών αριθμών

Ώρες γραφείου βοηθών

- Υπεύθυνη εργαστηρίων: Αντωνία Κυριακοπούλου
  - tonia@aueb.gr
  - Ώρες γραφείου: Πέπτη 13:00 – 15:00,
  - Εργαστήριο Επεξεργασίας Πληροφοριών, 4ος όροφος πτέρ. Αντωνιάδου.
- Υπεύθυνος εργασιών: Χρήστος Τσιλόπουλος
  - tsilochr@aueb.gr
  - Τετάρτη 13:00-15:00
  - MMlab, Κτίριο μεταπτυχιακού, Ευελπίδων 47 και Λευκάδος

Περίληψη

- Ειδικές Κατηγορίες Πινάκων
- Συνδεδεμένες λίστες
- Το πρόβλημα του Josephus
- Λίστες ή πίνακες;
- Επεξεργασία λιστών
- Συμβάσεις αρχής και τέλους
- Τάξη κυκλικής λίστας
- Διπλά συνδεδεμένη λίστα

Ειδικές Κατηγορίες Πινάκων

- Αραιοί (sparse) πίνακες: Πίνακες όπου μεγάλο ποσοστό των στοιχείων είναι 0 (π.χ. αραιοί γράφοι)
- Έστω ο πίνακας:

0	7	0	0
1	2	0	0
0	0	4	0
9	0	0	0

- Σπατάλη μνήμης η αποθήκευση ως 2διάστατο πίνακα
- Μπορούμε να χρησιμοποιήσουμε μονοδιάστατο πίνακα
  - Κάθε μη μηδενικό στοιχείο αποθηκεύεται ως μία τριάδα (συντεταγμένες στον πίνακα + τιμή)
  - 1 2 7 2 1 1 2 2 2 3 3 4 4 1 9

Ειδικές Κατηγορίες Πινάκων

- Τριγωνικοί πίνακες: Πίνακες όπου πάνω ή κάτω από τη διαγώνιο όλα τα στοιχεία είναι 0
- π.χ.:

0	0	0	0	
1	2	0	0	(κάτω τριγωνικός)
0	0	4	0	
9	0	0	0	

- Υπάρχουν πιο αποδοτικοί μέθοδοι αποθήκευσης με μονοδιάστατο πίνακα
- Η χρήση εναλλακτικών μεθόδων αποθήκευσης εξαρτάται από:
  - Πόσο σημαντική είναι η εξοικονόμηση μνήμης για την εφαρμογή
  - Τι άλλες λειτουργίες χρειάζεται να εκτελούμε στον πίνακα

Άλλες Υλοποιήσεις

- Αρκετές μέθοδοι για πίνακες είναι διαθέσιμες στην βιβλιοθήκη της Java, μέσω της κλάσης Vector
- Πιο ευέλικτη δομή από τους πίνακες (ένα αντικείμενο Vector μπορεί να αλλάζει μεγεθος)
- Προσπέλαση των στοιχείων είναι πιο αργή (μέσω της μεθόδου get())
- Για καλύτερη αποδοτικότητα και απλότητα του κώδικα, θα αποφεύγουμε να χρησιμοποιούμε έτοιμες μεθόδους

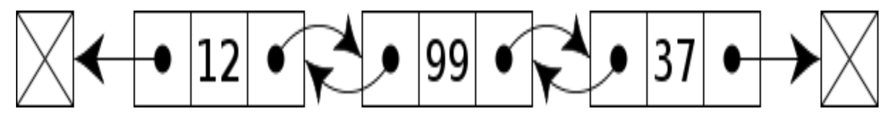
Συνδεδεμένες λίστες

- Συνδεδεμένη λίστα
  - Συλλογή δυναμικά κατανεμημένων κόμβων
  - Κάθε κόμβος περιέχει κάποια στοιχεία
  - Κάθε κόμβος περιέχει συνδέσεις προς άλλους
  - Ευμετάβλητη δομή με πολλές εφαρμογές
    - Απλή εισαγωγή και αφαίρεση στοιχείων
    - Δεν επιτρέπει τυχαία προσπέλαση κόμβων (π.χ. εύρεση του κ-οστού απαιτεί προσπέλαση των πρώτων κ στοιχείων)
  - Εναλλακτική λύση σε σχέση με πίνακα
    - Η καλύτερη λύση εξαρτάται από το πρόβλημα
- Τύποι συνδεδεμένων λιστών
  - Λίστες μονής σύνδεσης
  - Λίστες διπλής (ή πολλαπλής) σύνδεσης
  - Κυκλικές λίστες μονής σύνδεσης
  - Κυκλικές λίστες διπλής (ή πολλαπλής) σύνδεσης

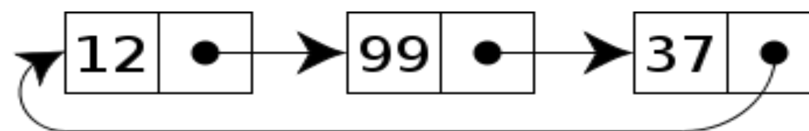
Συνδεδεμένες λίστες

- Λίστες μονής σύνδεσης

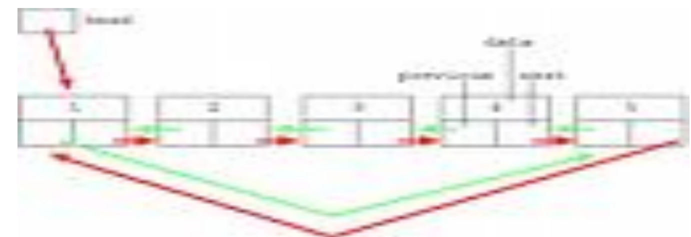
- Λίστες διπλής σύνδεσης



- Κυκλικές λίστες



- Κυκλικές λίστες διπλής σύνδεσης



Συνδεδεμένες λίστες

- Λίστα μονής σύνδεσης
  - Κάθε κόμβος περιέχει έναν κόμβο (ως σύνδεσμο προς τον επόμενό του)
  - Αυτοαναφορικές (self-referent) δομές
  - Παριστάνει μία ακολουθία στοιχείων
- Ορισμός κόμβου λίστας απλής σύνδεσης
  - Μπορεί να περιέχει οποιοδήποτε αντικείμενο

```
class Node{
  Object item;
  Node next;
  Node(Object v){
        item = v;
        next = null;
  } }
```

Καλή τεχνική να ανατίθενται αρχικές τιμές σε όλα τα στοιχεία

  - Δημιουργία ενός νέου κόμβου

```
Node x = new Node(j);
```

Συνδεδεμένες λίστες

- Διαγραφή από συνδεδεμένη λίστα
  - Αφαίρεση του κόμβου που ακολουθεί τον x

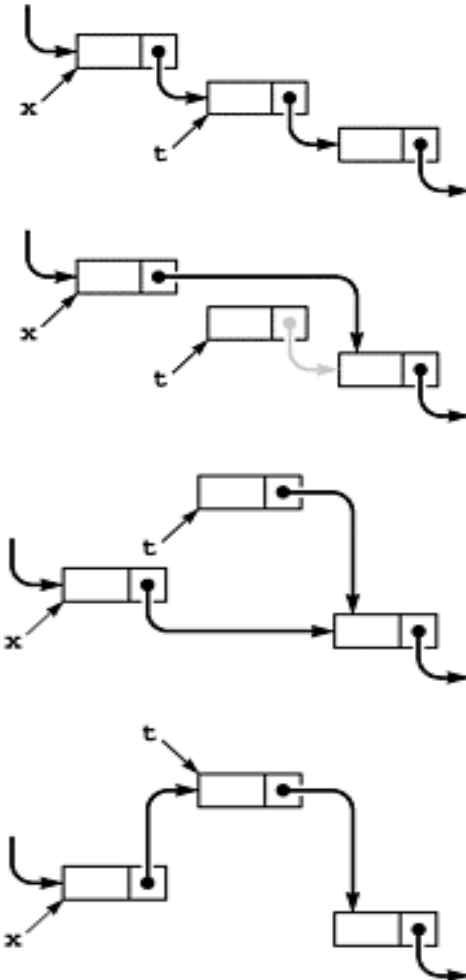
```
t = x.next;
x.next = t.next;
```

  - Εναλλακτικά

```
x.next = x.next.next
```

- Εισαγωγή σε συνδεδεμένη λίστα
  - Εισαγωγή του κόμβου t μετά τον x

```
t.next = x.next;
x.next = t;
```



Συνδεδεμένες λίστες

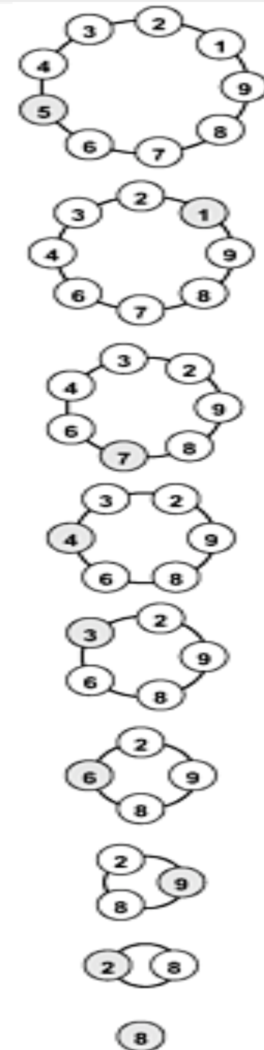
- Εντοπισμός του n-οστού κόμβου
  - Δεν υπάρχει άμεση μέθοδος
  - Αναγκαστικά διατρέχουμε τη λίστα
  - Π.χ. αν το `x` δείχνει στην κεφαλή της λίστας, τότε:

```
for (int i = 1; i < n;
i++)
     x = x.next;
```

- Συνηθισμένα λάθη κατά την επεξεργασία λιστών
  - Χρήση αόριστης αναφοράς (προς κανένα αντικείμενο)
  - Χρήση αναφοράς προς λάθος αντικείμενο

Το πρόβλημα του Josephus

- Συνάρτηση του Josephus
  - N άτομα τοποθετούνται σε έναν κύκλο
  - Αφαιρούμε το M-οστό άτομο (ξεκινώντας από το 1)
  - Ξεκινώντας από M+1 αφαιρούμε πάλι το M-οστό άτομο
  - κ.ο.κ. μέχρι να μείνει μόνο ένα άτομο
  - Josephus(N,M): το άτομο που θα μείνει στο τέλος
- Άμεση λύση με χρήση κυκλικής λίστας
  - Κατασκευή λίστας N κόμβων
  - Ο τελευταίος κόμβος δείχνει στον πρώτο
  - Διατρέχουμε τη λίστα μέχρι να αδειάσει
    - Κυκλική διάσχιση χωρίς ειδικό κώδικα
  - Διαγράφουμε το M-οστό στοιχείο κάθε φορά
    - Εύκολη αφαίρεση στοιχείων ακολουθώντας δείκτες



Το πρόβλημα του Josephus

```
class Josephus {
  static class Node {
        int val; Node next;
        Node(int v) { val = v; } }
  public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int M = Integer.parseInt(args[1]);
        Node t = new Node(1); Node x = t;
        for (int i = 2; i <= N; i++) {
                x.next = new Node(i)); x = x.next}
        x.next = t;//τελευταίος κόμβος δείχνει την αρχή
        while (x != x.next) {
                for (int i = 1; i < M; i++)
                        x = x.next;
                x.next = x.next.next; }
  Out.println("Survivor is " + x.val); } }
```

Υλοποίηση του Josephus με πίνακες

- Γενικά μπορούμε να υλοποιούμε λίστες με πίνακες (δεν είναι πάντα βολικό όμως)

- Χρειαζόμαστε 2 πίνακες:
  - val[i]: στοιχείο κόμβου i
  - next[i]: δείκτης επόμενου κόμβου
  - Διαγραφή κόμβου γίνεται με ενημέρωση του `next[]`
  - `next[x] = next[next[x]]`

	0	1	2	3	4	5	6	7	8
val	1	2	3	4	5	6	7	8	9
next	1	2	3	4	5	6	7	8	0
5	1	2	3	4	5	6	7	8	9
	1	2	3	5	5	6	7	8	0
1	1	2	3	4	5	6	7	8	9
	1	2	3	5	5	6	7	8	1
7	1	2	3	4	5	6	7	8	9
	1	2	3	5	5	7	7	8	1
4	1	2	3	4	5	6	7	8	9
	1	2	5	5	5	7	7	8	1
3	1	2	3	4	5	6	7	8	9
	1	5	5	5	5	7	7	8	1
6	1	2	3	4	5	6	7	8	9
	1	7	5	5	5	7	7	8	1
9	1	2	3	4	5	6	7	8	9
	1	7	5	5	5	7	7	1	1
2	1	2	3	4	5	6	7	8	9
	1	7	5	5	5	7	7	7	1

Λίστες ή πίνακες;

- Κατανάλωση χώρου
  - Πίνακας: σταθερός χώρος
  - Λίστα: χώρος ανάλογος των στοιχείων
    - Χρειάζεται χώρος και για τους δείκτες
- Χρόνος εισαγωγής / εξαγωγής
  - Πίνακας: πιθανή μετακίνηση στοιχείων
  - Λίστα: δεν χρειάζονται μετακινήσεις
- Χρόνος εύρεσης k-οστού στοιχείου
  - Πίνακας: κατευθείαν πρόσβαση `a[k]`
  - Λίστα: διατρέχουμε τους προηγούμενους κόμβους

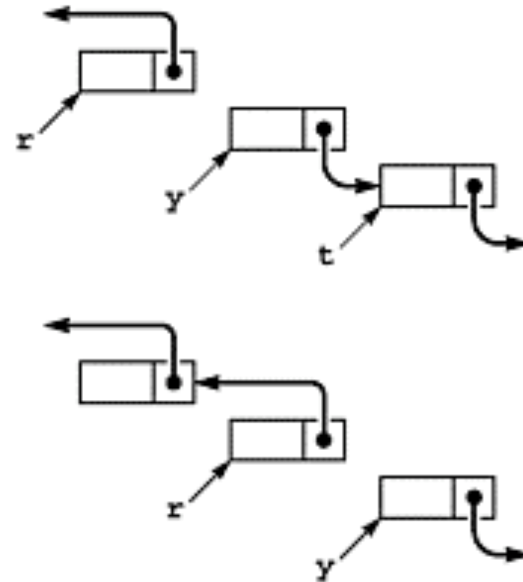
Επεξεργασία λιστών

- Παράδειγμα 1: Αντιστροφή λίστας μονής σύνδεσης
  - Είσοδος: ο δείκτης της κεφαλής μίας λίστας
  - Έξοδος: η ίδια λίστα με αντιστραμμένη σειρά στους κόμβους
  - Ιδέα: διατρέχω τη λίστα και διατηρώ 3 δείκτες
  - r: προηγούμενος κόμβος (που έχω ήδη επεξεργαστεί στην προηγούμενη επανάληψη),
  - y: τρέχον κόμβος,
  - t: επόμενος κόμβος

Επεξεργασία λιστών

- Αντιστροφή λίστας μονής σύνδεσης
  - r: προηγούμενος κόμβος, y: τρέχον κόμβος, t: επόμενος κόμβος

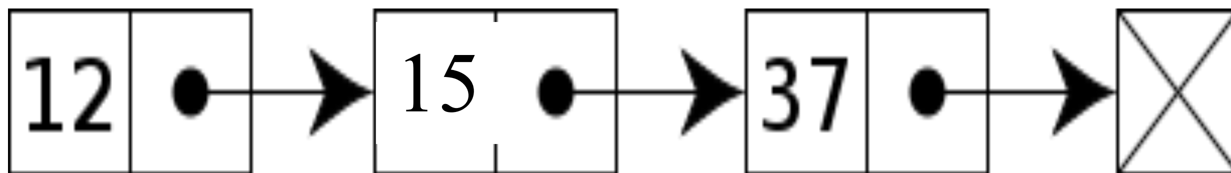
```
static Node reverse(Node x) {
  Node t, y = x, r = null;
  while (y != null) {
        t = y.next;
        y.next = r;
        r = y;
        y = t; }
  return r; }
```



Επεξεργασία λιστών

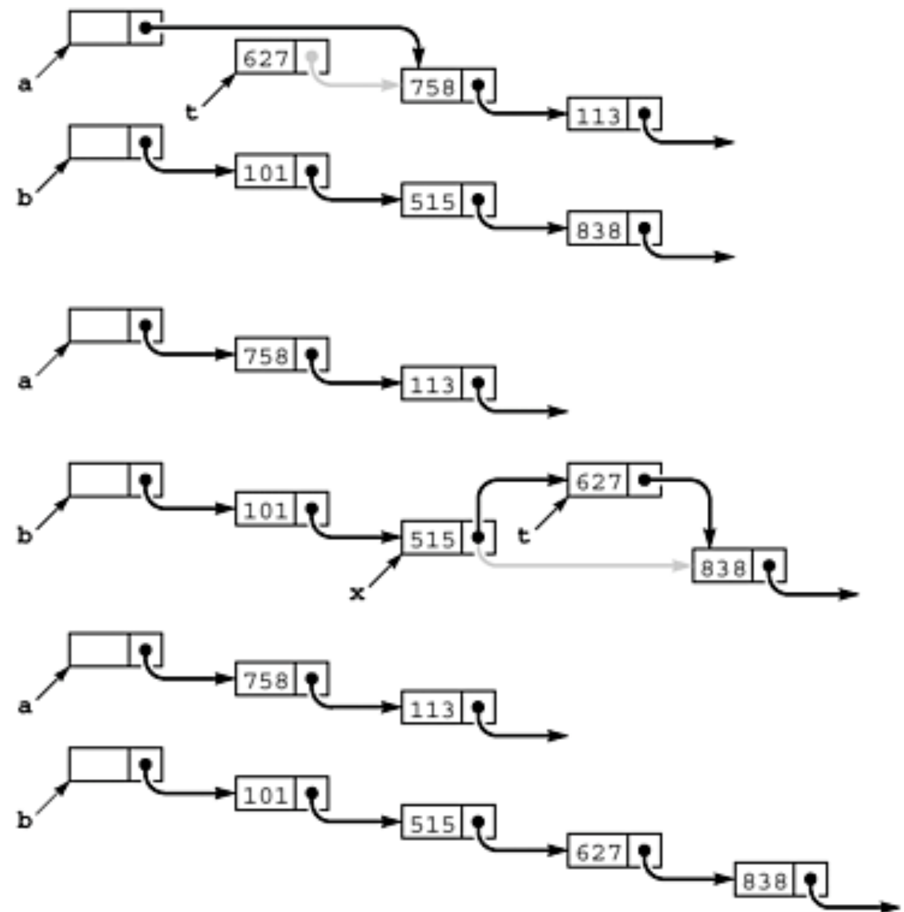
- Παράδειγμα 2: ταξινόμηση με εισαγωγή (insertion sort)
- Είσοδος: Ένα σύνολο ακεραίων που εισάγει ο χρήστης
- Έξοδος: Μία λίστα με τα ίδια στοιχεία αλλά ταξινομημένα σε αύξουσα σειρά
- π.χ. με είσοδο

37 12 15



High Level Idea

- Χρήση δύο λιστών
  - a: μη ταξινομημένη, δημιουργείται όταν διαβάζουμε την είσοδο
  - b: ταξινομημένη (αρχικά κενή)
- Μεταφορά κόμβων
  - Διατρέχουμε την a
  - Αφαιρούμε ένα στοιχείο
  - Βρίσκουμε τη σωστή θέση στη b
  - Εισάγουμε το στοιχείο
- Χρήση (ψευδο)-κόμβου κεφαλής
  - Βρίσκεται στην αρχή
  - Δεν περιέχει στοιχεία
  - Διευκολύνει την επεξεργασία



Επεξεργασία λιστών

- Ταξινόμηση με εισαγωγή (1/2)

```
class ListSortExample {
  static class Node {
        int val; Node next;
        Node(int v, Node t) { val = v; next = t; } }
  static Node create() {
        Node a = new Node(0, null);
        for (In.init(); !In.empty(); )
              a.next = new Node(In.getInt(), a.next);
        return a; }
static void print(Node h) {
        for (Node t = h.next; t != null; t = t.next)
                    Out.println(t.val + ""); }
… }
```

Επεξεργασία λιστών

- Ταξινόμηση με εισαγωγή (2/2)

```
class ListSortExample {
  …
  static Node sort(Node a) {
      Node t, u, x, b = new Node(0, null);
      while (a.next != null) {
          t = a.next; u = t.next; a.next = u;
          for (x = b; x.next != null; x = x.next)
              if (x.next.val > t.val) break;
              t.next = x.next;
          x.next = t; }
      return b; }
  public static void main(String[] args) {
  print(sort(create())); } }
```

Παρατηρήσεις

- Θα μπορούσαμε και με μία μόνο λίστα (Η a δεν είναι απαραίτητη). Όταν διαβάζουμε ένα στοιχείο, μπορούμε να το βάλουμε κατευθείαν στη σωστή θέση της b

- Σε άλλες εφαρμογές όμως ίσως χρειάζεται μία μέθοδος σαν την `create()` για να διαβάσουμε πρώτα όλη την είσοδο

- Πολυπλοκότητα?
  - Στη χειρότερη περίπτωση διατρέχουμε όλη την υπάρχουσα λίστα κάθε φορά

$$1 + 2 + \ldots + N-1 = N(N-1)/2 \text{ (τάξη μεγέθους } N^2\text{)}$$

Συμβάσεις αρχής και τέλους

Κυκλική, ποτέ κενή	
πρώτη εισαγωγή:	`head.next = head;`
εισαγωγή του t μετά το x:	`t.next = x.next; x.next=t;`
αφαίρεση μετά το x:	`x.next = x.next.next`

βρόχος διέλευσης:	`t = head;`
	`do {…t = t.next;}`
	`while (t != head)`

έλεγχος αν έχει ένα στοιχείο:	`if (head.next = head)`

- Σύμβαση 1η: κυκλική λίστα
  – Η λίστα δεν είναι ποτέ κενή
  – Το head δείχνει στην «αρχή» της λίστας

Συμβάσεις αρχής και τέλους

Αναφορά κεφαλής, null στην ουρά της λίστας	
ανάθεση αρχικών τιμών:	`head = null;`

εισαγωγή του t μετά το x:	`if (x == null) {`
	`  head = t; head.next = null;}`
	`else {`
	`  t.next = x.next; x.next = t; }`

αφαίρεση μετά το x:	`t = x.next; x.next = t.next;`
βρόχος διέλευσης:	`for (t=head; t!=null; t=t.next)`
έλεγχος αν είναι κενή:	`if (head == null)`

- Σύμβαση 2η: χωρίς ψευδο-κόμβους
  - Απλούστερη μορφή λίστας
  - Πιθανόν ειδικός κώδικας για αρχή και τέλος

Συμβάσεις αρχής και τέλους

Ψευδο-κόμβος κεφαλής, null στην ουρά της λίστας	
ανάθεση αρχικών τιμών:	`head = new Node();` `head.next = null;`
εισαγωγή του t μετά το x:	`t.next = x.next; x.next = t;`
αφαίρεση μετά το x:	`t = x.next; x.next = t.next;`
βρόχος διέλευσης:	`for (t=head.next; t!=null; t=t.next)`
έλεγχος αν είναι κενή:	`if (head.next == null)`

- Σύμβαση 3η: ψευδο-κόμβος κεφαλής
  - Δεν χρειάζεται ειδικός κώδικας στην εισαγωγή
  - Υπάρχει πάντα κάποιος κόμβος στη λίστα

Συμβάσεις αρχής και τέλους

Ψευδο-κόμβοι κεφαλής και ουράς	

ανάθεση αρχικών τιμών:	`head = new Node();`
	`z = new Node();`
	`head.next = z;`
	`z.next = z;`

εισαγωγή του t μετά το x:	`t.next = x.next; x.next = t;`
αφαίρεση μετά το x:	`x.next = x.next.next;`
βρόχος διέλευσης:	`for (t=head.next; t!=z; t=t.next)`
έλεγχος αν είναι κενή:	`if (head.next == z)`

- Σύμβαση 4η: ψευδο-κόμβος κεφαλής και ουράς
  - Τουλάχιστον δύο κόμβοι από τη στιγμή της δημιουργίας και έπειτα
  - Ο τελευταίος κόμβος δείχνει στον εαυτό του

Παρατηρήσεις

1. Το τι είδους λίστα ή σύμβαση θα χρησιμοποιήσετε εξαρτάται πάντα από το πρόβλημα
2. Συλλογή σκουπιδιών
   - Στη Java δεν χρειάζεται να απελευθερώνουμε τη μνήμη όταν σταματάμε να χρησιμοποιούμε κάποιο δείκτη. Γίνεται αυτόματα από τον Garbage Collector
   - Σε άλλες γλώσσες πρέπει να γίνει ρητά (στη C++ καλώντας την `delete`).
   - Προσοχή στη χρήση της μνήμης!
3. Για κάθε τύπο λίστας, μπορούμε να ορίσουμε μία κλάση που να περιέχει όλες τις χρήσιμες μεθόδους ως μέλη
   - Κάνει πιο εύκολο τον κώδικα για προγράμματα-πελάτες. Οι έλεγχοι γίνονται από τις μεθόδους της κλάσης

Παράδειγμα: Τάξη κυκλικής λίστας

- Όλες οι λειτουργίες υλοποιούνται από την τάξη

```
class CircularList {
  static class Node {
      int val; Node next;
      Node(int v) { val = v; }
  }
  Node next(Node x) { return x.next; }
  int val(Node x) { return x.val; }
  Node insert(Node x, int v) {
      Node t = new Node(v);
      if (x == null) t.next = t;
      else { t.next = x.next; x.next = t; }
      return t; }
  void remove(Node x) { x.next = x.next.next; }
}
```

Τάξη κυκλικής λίστας

- Συνάρτηση του Josephus με την τάξη κυκλικής λίστας
  - Πολύ απλούστερος κώδικας χάρη στις έτοιμες μεθόδους

```
class JosephusY {
  public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int M = Integer.parseInt(args[1]);
        CircularList L = new CircularList();
        CircularList.Node x = null;
        for (int i = 1; i <= N; i++)
                x = L.insert(x, i);
        while (x != L.next(x)) {
                for (int i = 1; i < M; i++)
                        x = L.next(x);
                L.remove(x); }
  Out.println("Survivor is " + L.val(x)); } }
```