ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΛΙΣΤΕΣ
4.9 Άλλες παραλλαγές Συνδεδεμένων Λιστών

Παραλλαγές συνδεδεμένων λιστών

Οι στοίβες και οι ουρές είναι ειδικές περιπτώσεις λιστών και είδαμε πώς μπορούν να υλοποιηθούν ως συνδεδεμένες λίστες.

Στην παράγραφο αυτήν θα δούμε ακόμα δύο περιπτώσεις λιστών:

- τις **λίστες με κόμβους κεφαλή** (lists with head nodes) και
- τις **κυκλικές συνδεδεμένες λίστες** (circular linked lists).

Λίστες με κόμβους κεφαλή

Ο πρώτος κόμβος μιας τυπικής συνδεδεμένης λίστας διαφέρει από τους υπόλοιπους, γιατί δεν έχει προηγούμενο κόμβο.

Εξ αιτίας αυτού του γεγονότος ξεχωρίσαμε δύο περιπτώσεις για τις διαδικασίες εισαγωγής και διαγραφής.

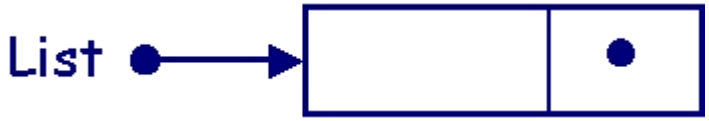
Κάτι τέτοιο όμως μπορεί να αποφευχθεί αν εξασφαλίσουμε ότι κάθε κόμβος που περιέχει κάποιο στοιχείο θα έχει προηγούμενο κόμβο, εισάγοντας στην αρχή της λίστας έναν εικονικό πρώτο κόμβο, τον **κόμβο κεφαλή** (**head node**):

- Στο τμήμα δεδομένου αυτού του κόμβου δεν αποθηκεύεται στην πραγματικότητα κανένα στοιχείο της λίστας.
- Ο κόμβος κεφαλή είναι ο προηγούμενος του κόμβου στον οποίο αποθηκεύεται το πρώτο στοιχείο, γιατί δείχνει σ' αυτόν τον πραγματικά πρώτο κόμβο.

Ένα παράδειγμα φαίνεται παρακάτω:

Λίστες με κόμβους κεφαλή

Σ' αυτού του είδους τις λίστες κάθε συνδεδεμένη λίστα πρέπει να έχει έναν κόμβο κεφαλή κι επομένως μια κενή λίστα έχει μόνο τον κόμβο κεφαλή, όπως φαίνεται παρακάτω:



Επομένως, για να δημιουργήσουμε μια κενή λίστα, δεν χρειάζεται απλά να δώσουμε την τιμή NULL σε έναν δείκτη List, αλλά πρέπει να πάρουμε έναν κόμβο κεφαλή στον οποίο να δείχνει ο List και το πεδίο δεσμού του να είναι NULL. Στην C αυτό γίνεται με τις ακόλουθες εντολές:

```
List = (ListPointer)malloc(sizeof(struct ListNode);
List->Next = NULL;
```

Για να εξετάσουμε τώρα αν μια τέτοια λίστα είναι κενή αρκεί να ελέγξουμε αν List->Next == NULL και όχι αν List == NULL.

Λίστες με κόμβους κεφαλή

Όπως φαίνεται και από το προηγούμενο σχήμα, δημιουργήσαμε μια κενή λίστα με έναν κόμβο κεφαλή ορίζοντας το τμήμα δεσμού του σε NULL χωρίς όμως να δίνουμε κάποια τιμή στο τμήμα δεδομένου του.

Σε ορισμένες περιπτώσεις είναι δυνατό να αποθηκεύουμε στο τμήμα δεδομένου του κόμβου κεφαλή κάποια πληροφορία σχετική με τη λίστα.

Αν, για παράδειγμα οι *Ελένη*, *Μαρία* και *Φωτεινή* δουλεύουν στην εταιρεία *Star*, τότε μπορούμε να αποθηκεύσουμε το όνομα της εταιρείας στον κόμβο κεφαλή ως εξής:

Επειδή σε μια λίστα με κόμβο κεφαλή, για όλους τους κόμβους που περιέχουν στοιχεία της λίστας, υπάρχει προηγούμενος κόμβος, οι διαδικασίες εισαγωγής και διαγραφής είναι πιο απλοποιημένες.

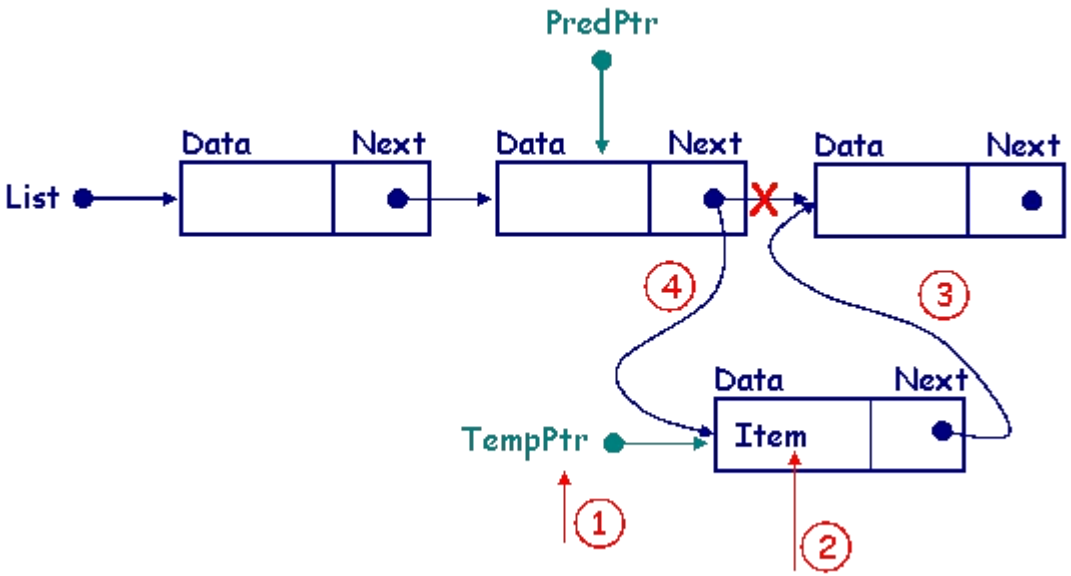
Διαδικασία εισαγωγής στοιχείου

(1) Παίρνουμε ένα νέο κόμβο, στον οποίο δείχνει προσωρινά ο δείκτης *TempPtr*

(2) Data(*TempPtr*) ← *Item*

(3) Next(*TempPtr*) ← Next(*PredPtr*)

(4) Next(*PredPtr*) ←*TempPtr*



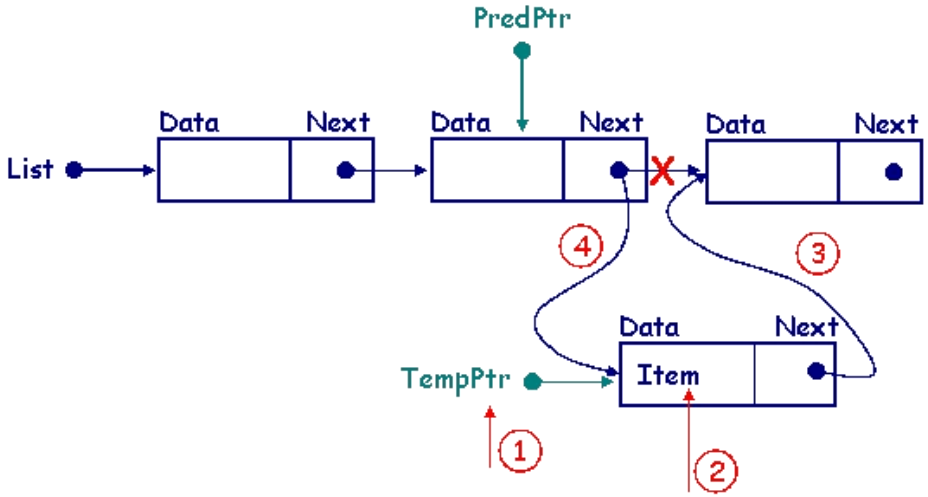
Διαδικασία εισαγωγής στοιχείου

/*Δέχεται: Μια συνδεδεμένη λίστα με κόμβο κεφαλή, που δεικτοδοτείται από τον *List*, ένα στοιχείο δεδομένων *Item* και έναν δείκτη *PredPtr*.

Λειτουργία: Εισάγει έναν κόμβο, που περιέχει το *Item,* μέσα στην συνδεδεμένη λίστα μετά από τον κόμβο που δεικτοδοτείται από τον *PredPtr*.

Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα με κόμβο κεφαλή που δεικτοδοτείται από τον *List*.*/

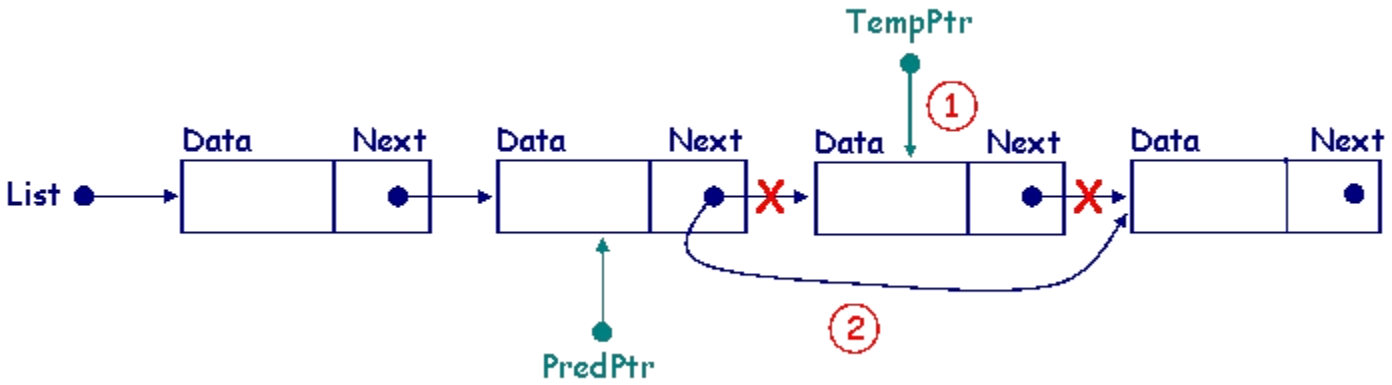
1. Πάρε έναν κόμβο που να δεικτοδοτείται από τον *TempPtr*
2. Data(*TempPtr*) ←*Item*
3. Next(*TempPtr*) ← Next(*PredPtr*)
4. Next(*PredPtr*) ← *TempPtr*



Διαδικασία διαγραφής στοιχείου

(1) TempPtr ← Next(PredPtr)

(2) Next(PredPtr) ← Next(TempPtr)



Αλγόριθμος διαγραφής στοιχείου

/*Δέχεται: Μια συνδεδεμένη λίστα με κόμβο κεφαλή που δεικτοδοτείται από τον *List* και έναν δείκτη *PredPtr*.

Λειτουργία: Διαγράφει από τη συνδεδεμένη λίστα τον κόμβο που έχει για προηγούμενό του αυτόν στον οποίο δείχνει ο *PredPtr*, αν η λίστα δεν είναι κενή.

Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο να δεικτοδοτείται από τον *List*.

Έξοδος: Ένα μήνυμα κενής λίστας αν η συνδεδεμένη λίστα είναι κενή.*/

Αν η λίστα είναι κενή **τότε**

Γράψε 'Προσπαθείς να διαγράψεις στοιχείο από κενή λίστα'

Αλλιώς

1. *TempPtr* ← Next(*PredPtr*)
2. Next(*PredPtr*) ← Next(*TempPtr*)
3. Να επιστρέψεις τον κόμβο στον οποίο δείχνει ο *TempPtr* στην δεξαμενή των διαθέσιμων κόμβων

Τέλος_αν

Αλγόριθμος διάσχισης

(*Δέχεται: Μια συνδεδεμένη λίστα με κόμβο κεφαλή που δεικτοδοτείται από τον *List*.

Λειτουργία: Διασχίζει τη λίστα με τη βοήθεια του δείκτη *CurrPtr* που δείχνει τον τρέχοντα κάθε φορά κόμβο της συνδεδεμένης λίστας και επεξεργάζεται κάθε στοιχείο μόνο μια φορά.

Επιστρέφει: Εξαρτάται από το είδος της επεξεργασίας.*)

1. *CurrPtr* ← Next(*List*)

2. **Όσο** *CurrPtr* != **NULL επανάλαβε**

    α. Πάρε το τρέχον Data(*CurrPtr*) για επεξεργασία

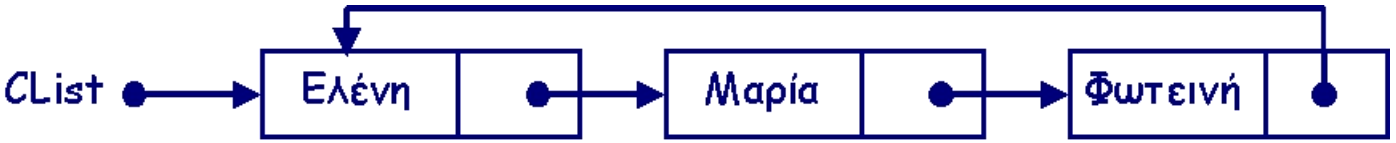
    β. *CurrPtr* ← Next(*CurrPtr*)

Τέλος_επανάληψης

Κυκλικές συνδεδεμένες λίστες

Όταν μελετήσαμε την υλοποίηση των ουρών με πίνακα, είδαμε ότι μπορούσαμε να αντιμετωπίσουμε το πρόβλημα της μετατόπισης των στοιχείων μέσα στον πίνακα, χρησιμοποιώντας έναν κυκλικό πίνακα στον οποίο το πρώτο στοιχείο ακολουθεί το τελευταίο.

Η ίδια ιδέα μπορεί να εφαρμοστεί και σε μια συνδεδεμένη λίστα, αν ο τελευταίος κόμβος δείχνει στον πρώτο, δηλαδή αν έχουμε μια **κυκλική συνδεδεμένη λίστα** (**circular linked list**) όπως η παρακάτω:



Κυκλικές συνδεδεμένες λίστες

Από το σχήμα φαίνεται ότι κάθε κόμβος έχει έναν προηγούμενο και έναν επόμενο, εκτός από την περίπτωση που η λίστα είναι κενή.

Επομένως, οι αλγόριθμοι εισαγωγής και διαγραφής στοιχείου μιας κυκλικής συνδεδεμένης λίστας είναι πιο απλοί από τους αντίστοιχους της τυπικής συνδεδεμένης λίστας, επειδή όλοι οι κόμβοι έχουν προηγούμενο, όμως τώρα **πρέπει να ξεχωρίσουμε την περίπτωση που εισάγουμε στοιχείο σε κενή λίστα ή διαγράφουμε το μοναδικό στοιχείο μιας λίστας και την αφήνουμε κενή**.

Τροποποιήσεις πρέπει να γίνουν και στον αλγόριθμο διάσχισης της λίστας.

Εισαγωγή στοιχείου σε κυκλική συνδεδεμένη λίστα

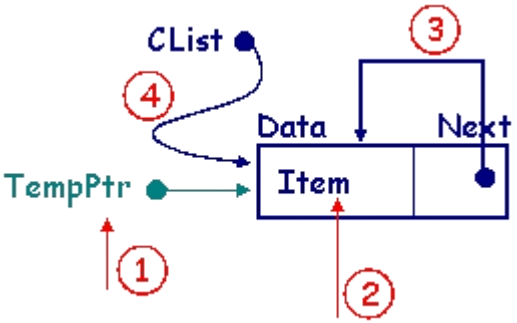
(1) Παίρνουμε ένα νέο κόμβο, στον οποίο δείχνει προσωρινά ο δείκτης *TempPtr*

(2) Data(*TempPtr*) ← *Item*

Αν **η λίστα είναι κενή**:

(3) Next(TempPtr) ← TempPtr

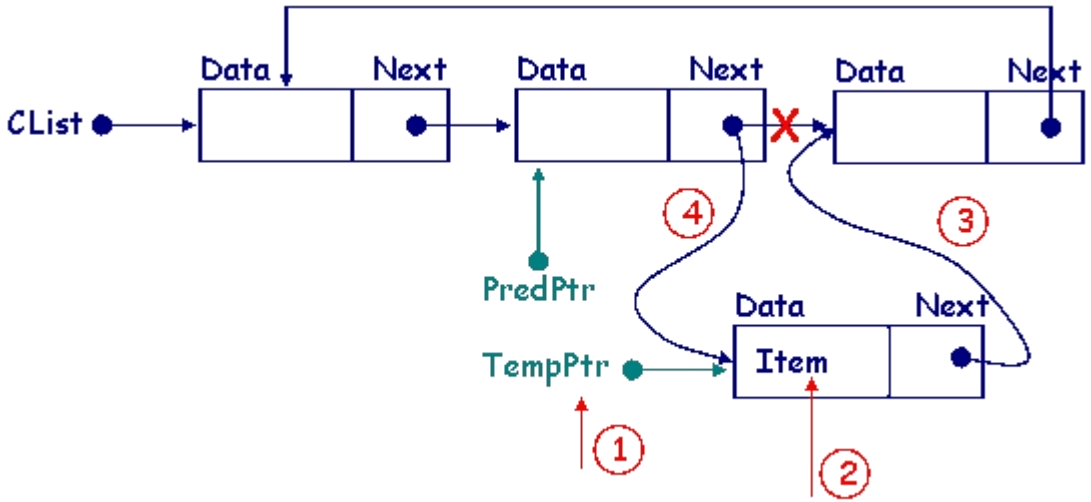
(4) CList ← TempPtr



Αν **η λίστα δεν είναι κενή**:

(3) Next(*TempPtr*) ← Next(*PredPtr*)

(4) Next(*PredPtr*) ← *TempPtr*



Αλγόριθμος εισαγωγής στοιχείου

/*Δέχεται: Μια κυκλική συνδεδεμένη λίστα με τον πρώτο κόμβο να δεικτοδοτείται από τον *CList*, ένα στοιχείο *Item* και έναν δείκτη *PredPtr*.

Λειτουργία: Εισάγει έναν κόμβο, που περιέχει το *Item*, μέσα στην κυκλική λίστα μετά από τον κόμβο που δεικτοδοτείται από τον *PredPtr* (εφόσον υπάρχει κάποιος).

Επιστρέφει: Την τροποποιημένη κυκλική συνδεδεμένη λίστα που δεικτοδοτείται από τον *CList*.*/

1. Πάρε έναν κόμβο στον οποίο να δείχνει ο *TempPtr*
2. *Data*(*TempPtr*) ← *Item*
3. **Αν** η λίστα είναι κενή **τότε**
   α. Next(*TempPtr*) ← *TempPtr*
   β. *CList* ← *TempPtr*

   Αλλιώς
   α. Next(*TempPtr*) ← Next(*PredPtr*)
   β. Next(*PredPtr*) ← *TempPtr*

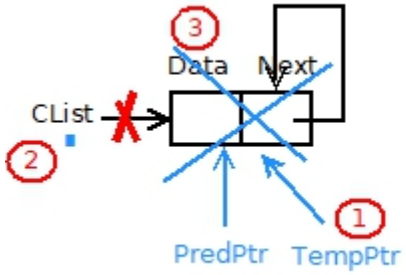
   Τέλος_αν

Διαγραφή στοιχείου από κυκλική συνδεδεμένη λίστα

(1) *TempPtr* ← Next(*PredPtr*)

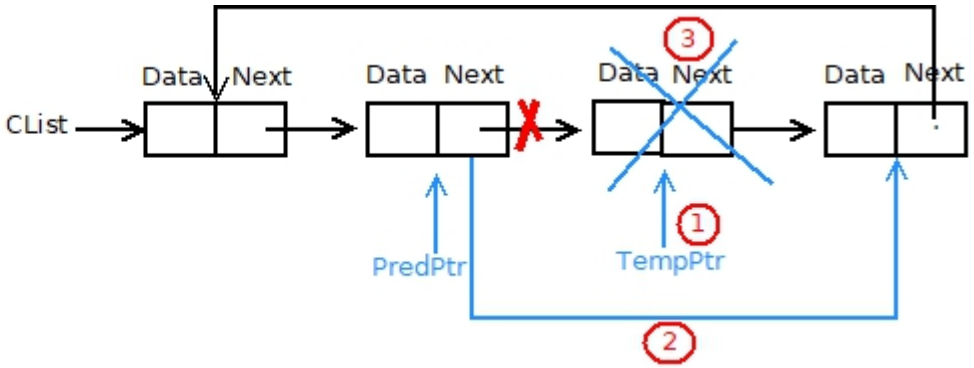
Αν *TempPtr* == *PredPtr* **τότε:**

(2) *CList* ← NULL



Αν *TempPtr* != *PredPtr*:

(2) Next(*PredPtr*) ← Next(*TempPtr*)



(3) Επέστρεψε τον κόμβο στον οποίο δείχνει ο *TempPtr* στη δεξαμενή των διαθέσιμων κόμβων

Αλγόριθμος διαγραφής στοιχείου

/*Δέχεται: Μια συνδεδεμένη κυκλική λίστα με τον πρώτο κόμβο να δεικτοδοτείται από τον *CList* και έναν δείκτη *PredPtr*.

Λειτουργία: Διαγράφει από τη λίστα τον κόμβο που έχει για προηγούμενό του αυτόν στον οποίο δείχνει ο *PredPtr*, αν υπάρχει κάποιος.

Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο να δεικτοδοτείται από τον *CList*.*/

Αν η λίστα είναι κενή **τότε**

Γράψε 'Προσπαθείς να διαγράψεις στοιχείο από κενή λίστα'

Αλλιώς

1. *TempPtr* ← Next(*PredPtr*)
2. **Αν** *TempPtr* == *PredPtr* **τότε**

CList ← NULL

Αλλιώς

Next(*PredPtr*) ← Next(*TempPtr*)

Τέλος_αν

3. Να επιστρέψεις τον κόμβο στον οποίο δείχνει ο *TempPtr* στη δεξαμενή των διαθέσιμων κόμβων

Τέλος_αν

Αλγόριθμος διάσχισης κυκλ. Συνδεδεμένης λίστας

/*Δέχεται: Μια κυκλική συνδεδεμένη λίστα με τον πρώτο κόμβο να δεικτοδοτείται από τον *CList*.

Λειτουργία: Διασχίζει την κυκλική συνδεδεμένη λίστα με τη βοήθεια του δείκτη *CurrPtr* που δείχνει τον τρέχοντα κάθε φορά κόμβο της κυκλικής συνδεδεμένης λίστας και επεξεργάζεται κάθε στοιχείο της κυκλικής συνδεδεμένης λίστας μόνο μια φορά.

Επιστρέφει: Εξαρτάται από το είδος της επεξεργασίας.*/

Αν η λίστα δεν είναι κενή **τότε**

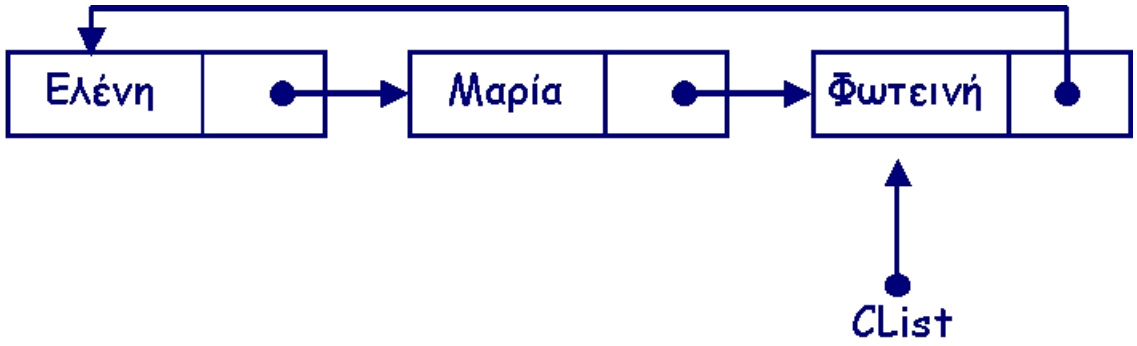
1. CurrPtr ← CList
2. **Αρχή_επανάληψης**
   - α. Πάρε το τρέχον στοιχείο *Data*(*CurrPtr*) για επεξεργασία
   - β. *CurrPtr* ← Next(*CurrPtr*)

   Μέχρις_ότου (*CurrPtr* == *CList*)

Τέλος_αν

Δεικτοδότηση τελευταίου στοιχείου σε κυκλική ΣΛ

Σε ορισμένες περιπτώσεις είναι προτιμότερο ο δείκτης Clist να δείχνει στον τελευταίο κόμβο και όχι στον πρώτο, γιατί έτσι μπορούμε να έχουμε άμεση πρόσβαση στον τελευταίο κόμβο και σχεδόν άμεση πρόσβαση και στον πρώτο, αφού ο Next(Clist) δείχνει στον πρώτο κόμβο:



Μπορούμε, επίσης, να έχουμε μια κυκλική συνδεδεμένη λίστα με κόμβο κεφαλή, όπως φαίνεται στο παρακάτω σχήμα: