Δομές Δεδομένων

Ενότητα 7: Άλλες παραλλαγές Συνδεδεμένων Λιστών-Παράσταση Αραιού Πολυωνύμου με Συνδεδεμένη Λίστα

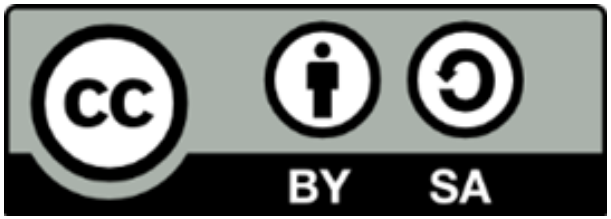
Καθηγήτρια Μαρία Σατρατζέμη

Τμήμα Εφαρμοσμένης Πληροφορικής

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοποί ενότητας

- Να γνωρίζουν τις παραλλαγές συνδεδεμένων λιστών
- Τις λίστες με κόμβο κεφαλή και τους αλγόριθμους εισαγωγής, διαγραφής στοιχείου και διάσχισης
- Τις κυκλικές λίστες και τους αλγόριθμους εισαγωγής, διαγραφής στοιχείου και διάσχισης
- Να χρησιμοποιούν συνδεδεμένες λίστες (με κόμβο κεφαλή) για να αποθηκεύσουν αραιά πολυώνυμα.
- Υλοποίηση της πρόσθεσης αραιών πολυωνύμων με κόμβο κεφαλή

Περιεχόμενα ενότητας[1]

- Παραλλαγές συνδεδεμένων λιστών
- Λίστες με κόμβους κεφαλή
  - Αλγόριθμος Εισαγωγή στοιχείου
  - Αλγόριθμος διαγραφής στοιχείου
  - Αλγόριθμος διάσχισης εισαγωγής στοιχείου
- Κυκλικές συνδεδεμένες λίστες
  - Εισαγωγή στοιχείου σε κυκλική συνδεδεμένη λίστα
  - Αλγόριθμος διαγραφής στοιχείου

Περιεχόμενα ενότητας[2]

- Πολυώνυμα
- Αραιά πολυώνυμα
- Αραιά πολυώνυμα: αποθήκευση λίστας
- Δηλώσεις
- Παράδειγμα: πρόσθεση πολυωνύμων
- Αλγόριθμος πρόσθεσης πολυωνύμων
- Εφαρμογή του αλγορίθμου
- Διαδικασία πρόσθεσης συνδεδεμένων πολυωνύμων

Άλλες παραλλαγές Συνδεδεμένων Λιστών

Παραλλαγές συνδεδεμένων λιστών

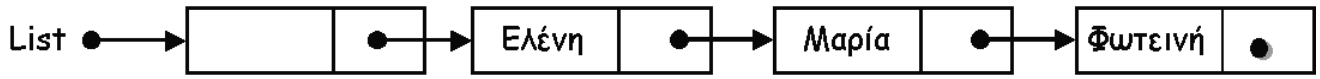
- Οι στοίβες και οι ουρές είναι ειδικές περιπτώσεις λιστών και είδαμε πώς μπορούν να υλοποιηθούν ως συνδεδεμένες λίστες.
- Στην παράγραφο αυτήν θα δούμε ακόμα δύο περιπτώσεις λιστών:
- τις λίστες με κόμβους κεφαλή (lists with head nodes) και
- τις κυκλικές συνδεδεμένες λίστες (circular linked lists).

Λίστες με κόμβους κεφαλή -1-

- 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):

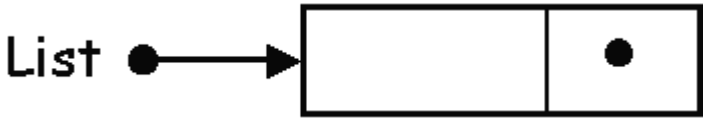
Λίστες με κόμβους κεφαλή -2-

- 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 -3-

- 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:



- 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;
```

Λίστες με κόμβους κεφαλή -4-

- 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 -5-

- 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.

Λίστες με κόμβους κεφαλή -6-

- 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.

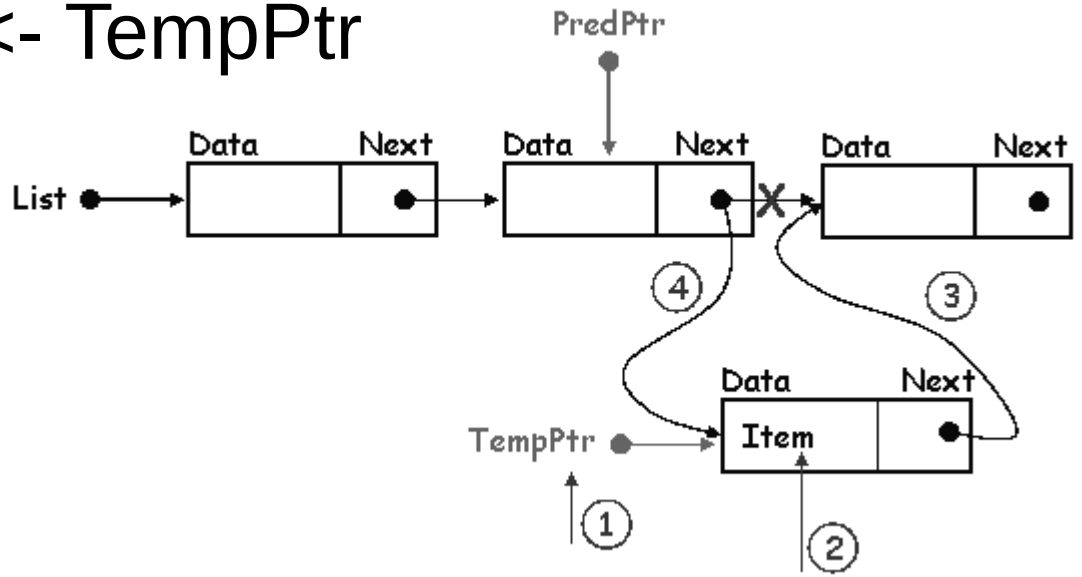
Διαδικασία εισαγωγής στοιχείου

(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



Αλγόριθμος 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.

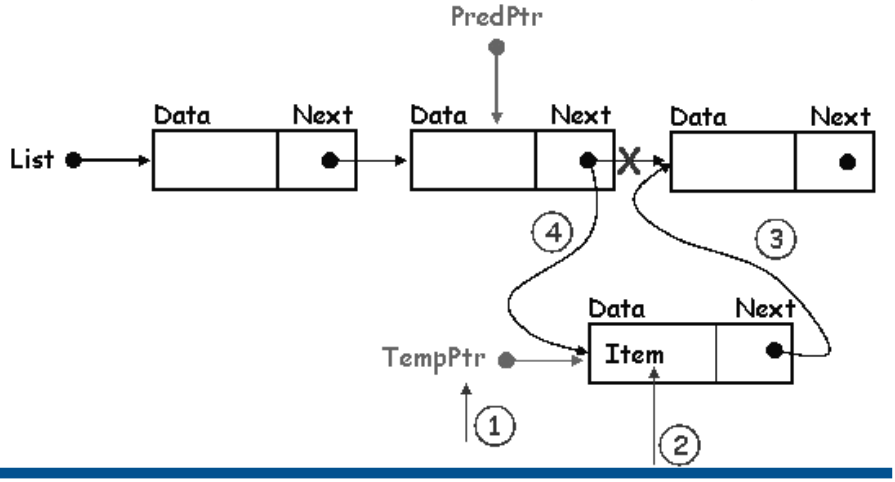
Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα με κόμβο κεφαλή που δεικτοδοτείται από τον List.*/

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

(2) Data(TempPtr) <-Item

(3) Next(TempPtr) <- Next(PredPtr)

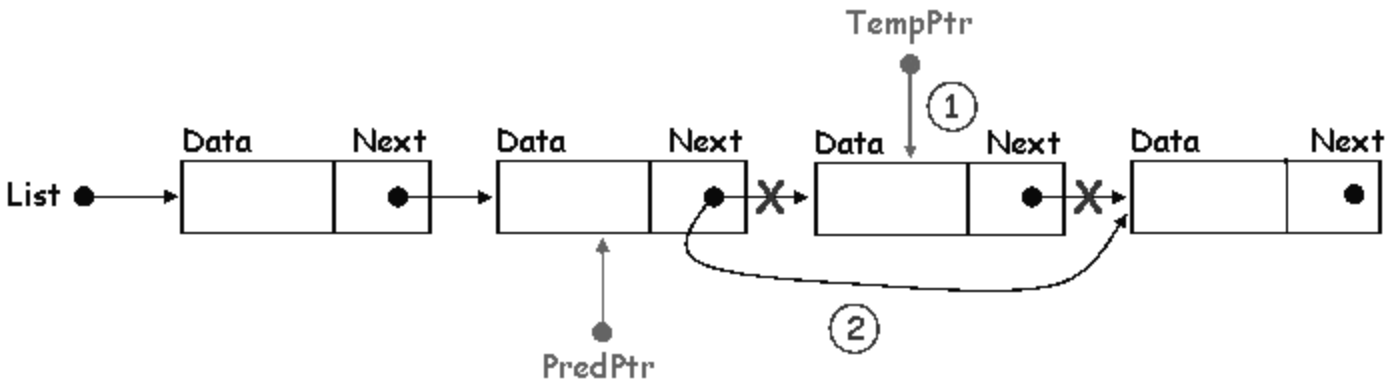
(4) Next(PredPtr)<- TempPtr



Διαδικασία διαγραφής στοιχείου

(1) TempPtr <- Next(PredPtr)

(2) Next(PredPtr)<-Next(TempPtr)



Αλγόριθμος διαγραφής στοιχείου -1-

/*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.*/

Αλγόριθμος διαγραφής στοιχείου -1-

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 -1-

/*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.*/

Αλγόριθμος διάσχισης -2-

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 -1-

- 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.

Κυκλικές συνδεδεμένες λίστες -2-

- 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

Εισαγωγή στοιχείου σε κυκλική συνδεδεμένη λίστα

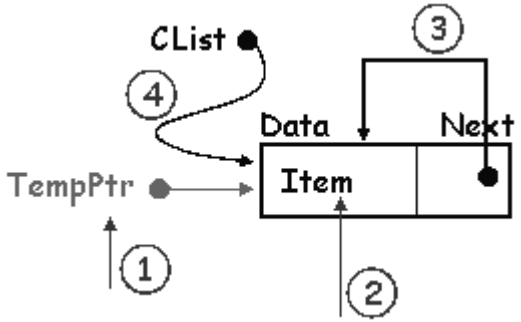
(1) Παίρνουμε ένα νέο κόμβο, στον οποίο δείχνει προσωρινά ο δείκτης TempPtr

(2) Data(TempPtr) <-Item

Αν η λίστα είναι κενή:

(3) Next(TempPtr) <-TempPtr

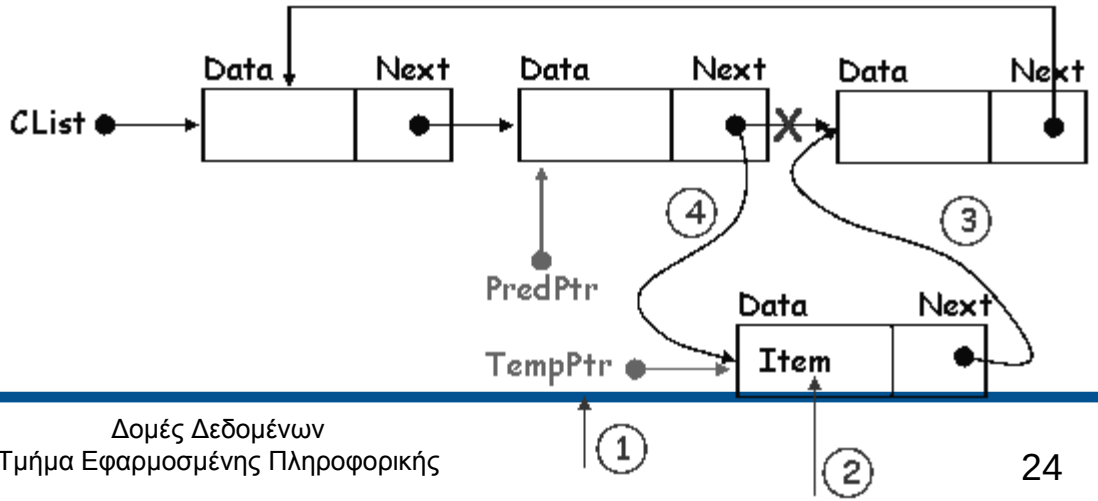
(4) CList <- TempPtr



Αν η λίστα δεν είναι κενή:

(3) Next(TempPtr) <- Next(PredPtr)

(4) Next(PredPtr) <- TempPtr



Αλγόριθμος εισαγωγής στοιχείου -1-

/*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.*/

Αλγόριθμος εισαγωγής στοιχείου -2-

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

Αλγόριθμος διαγραφής στοιχείου -1-

/*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.*/

Αλγόριθμος διαγραφής στοιχείου -2-

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 ← nil
        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 -1-

/*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.*/

Αλγόριθμος διάσχισης κυκλ. Συνδεδεμένης λίστας -2-

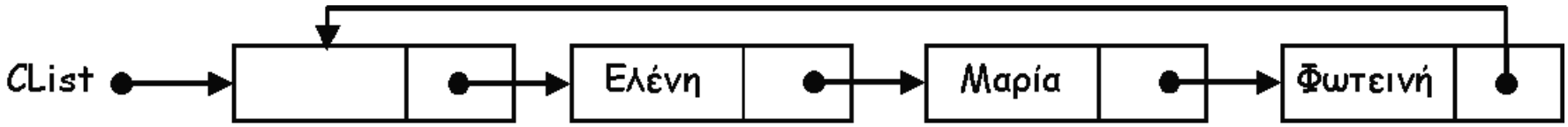
Αν η λίστα δεν είναι κενή τότε

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

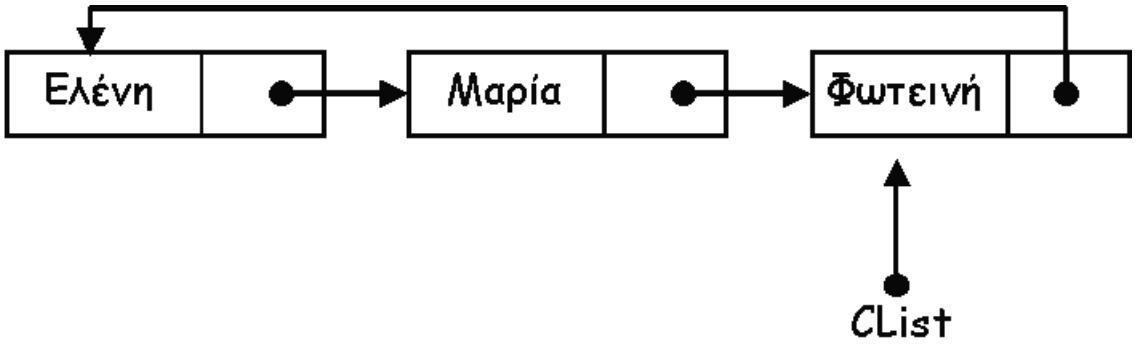
Αλγόριθμος διαγραφής στοιχείου σε κυκλ. ΣΛ. -1-

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



Αλγόριθμος διαγραφής στοιχείου σε κυκλ. ΣΛ. -2-

- 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:



Παράσταση Αραιού Πολυωνύμου με Συνδεδεμένη Λίστα

Πολυώνυμα

- Είναι γνωστό ότι σ' ένα πολυώνυμο P(x) της μορφής

  $$P(x)= a^0 + a^1x + a^2x^2 + ... + a^nx^n$$

  οι a^0,, a^1, a^2, ... , a^n ονομάζονται συντελεστές (coefficients) του πολυωνύμου και ο αριθμός n που αντιστοιχεί στη μεγαλύτερη δύναμη του x με $a^n \neq 0$ ονομάζεται βαθμός (degree) του πολυωνύμου.
- Έτσι, για παράδειγμα, το πολυώνυμο

  $$P(x)= 3 - 5x + 21x^2 + x^3$$

  έχει συντελεστές 3, -5, 21, 1 και βαθμό 3, ενώ το πολυώνυμο $Q(x)= 5$

  έχει έναν συντελεστή, 5, και βαθμό 0.

Αραιά πολυώνυμα -1-

- Αν ο βαθμός του πολυωνύμου που αποθηκεύεται σε έναν τέτοιο πίνακα δεν διαφέρει πολύ από το άνω όριο που τίθεται από το μέγεθος του πίνακα και οι μηδενικοί συντελεστές δεν είναι πολλοί, τότε μια αναπαράσταση σαν την παραπάνω είναι ικανοποιητική.
- Ωστόσο, υπάρχουν πολυώνυμα που έχουν λίγους μη μηδενικούς συντελεστές. Τα πολυώνυμα αυτά ονομάζονται **αραιά πολυώνυμα** (**sparse polynomials**) και ένας πίνακας δεν είναι η κατάλληλη αποθηκευτική δομή για τους συντελεστές τους.

Αραιά πολυώνυμα -2-

- Ένα παράδειγμα αραιού πολυωνύμου είναι το παρακάτω:

  $P(x)= 12x - 3x^2 + 3x^{70}$

  το οποίο μπορεί να γραφτεί και ως

  $P(x) = 0 + 12x - 3x^2 + 0x^3 + 0x^4 + ... + 0x^{69} + 3x^{70}$.

Αραιά πολυώνυμα -3-

- Η αποθήκευση αυτού του πολυωνύμου σε πίνακα θα απαιτούσε 71 θέσεις, από τις οποίες μόνο οι 3 θα είχαν μη μηδενικές τιμές, ενώ οι υπόλοιπες 68 θα είχαν μηδενικές.
- Κάτι τέτοιο συνεπάγεται σπατάλη μνήμης, την οποία μπορούμε να αποφύγουμε αν αποθηκεύσουμε μόνο τους μη μηδενικούς συντελεστές.
- Βέβαια, σ' αυτήν την περίπτωση θα πρέπει να αποθηκεύσουμε και τη δύναμη του x που αντιστοιχεί σε κάθε συντελεστή.

Αραιά πολυώνυμα -5-

- Επομένως, μπορούμε να παραστήσουμε ένα πολυώνυμο σαν μια λίστα από ζεύγη συντελεστών-εκθετών.

- Για το παραπάνω πολυώνυμο η λίστα που σχηματίζεται είναι η ακόλουθη:

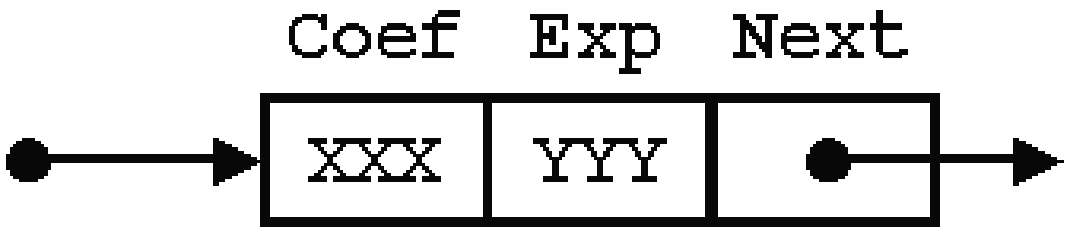
((12, 1),(-3, 2),(3, 70))

Αραιά πολυώνυμα: αποθήκευση λίστας -1-

- Για την αποθήκευση μιας τέτοιας λίστας μπορούμε να χρησιμοποιήσουμε έναν πίνακα εγγραφών με ένα πεδίο Coefficient για τον συντελεστή και ένα πεδίο Exponent για τον εκθέτη.
- Πάλι όμως θα έχουμε το πρόβλημα του σταθερού μεγέθους του πίνακα που περιορίζει το μέγεθος της λίστας.
- Κατά συνέπεια, μια πιο κατάλληλη δομή αποθήκευσης είναι η συνδεδεμένη λίστα.

Αραιά πολυώνυμα: αποθήκευση λίστας -2-

- Κάθε κόμβος της λίστας θα περιέχει:
- ένα τμήμα Coef, για την αποθήκευση ενός μη μηδενικού συντελεστή,
- ένα τμήμα Exp, για την αποθήκευση του αντίστοιχου εκθέτη, και
- ένα τμήμα Next, για την αποθήκευση του δείκτη που δείχνει στον επόμενο όρο του πολυωνύμου:

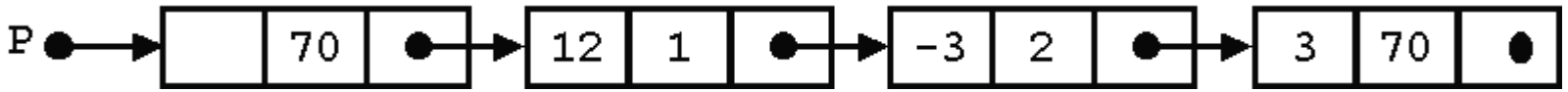


Αραιά πολυώνυμα: αποθήκευση λίστας -3-

- Για παράδειγμα, το πολυώνυμο

  $P(x)= 12x - 3x^2 + 3x^{70}$

- μπορεί να αποθηκευτεί σε μια συνδεδεμένη λίστα με κόμβο κεφαλή, όπου ο βαθμός του πολυωνύμου αποθηκεύεται στο τμήμα Exp του κόμβου κεφαλή:



Δηλώσεις

```
typedef int CoefficientType;
typedef  struct PolyNode *PolyPointer;
typedef  struct PolyNode {
        CoefficientType Coef;
        int  Expo;
        PolyPointer next;
} PolyNode;

typedef enum {
    FALSE, TRUE
} boolean;
```

Παράδειγμα: πρόσθεση πολυωνύμων -1-

- Για να γίνει κατανοητός ο τρόπος επεξεργασίας των πολυωνύμων, μπορούμε να θεωρήσουμε τη λειτουργία της πρόσθεσης δυο πολυωνύμων, π.χ. των P(x) και Q(x):

$$P(x)= 2 + 5x^2 - 12x^3 - x^4 - x^6$$

$$Q(x)= 12x^3 - x^4 + 2x^5 + 24x^6$$

- Για να προσθέσουμε δύο πολυώνυμα, προσθέτουμε τους συντελεστές των όρων που έχουν τον ίδιο εκθέτη. Το αποτέλεσμα της πρόσθεσης είναι:

$$W(x)= P(x) + Q(x)= 2 + 5x^2 - 2x^4 + 2x^5 + 23x^6$$

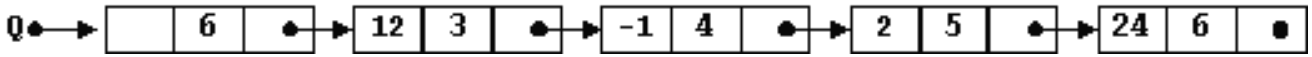
Παράδειγμα: πρόσθεση πολυωνύμων -2-

$P(x)= 2 + 5x^2 - 12x^3 - x^4 - x^6$

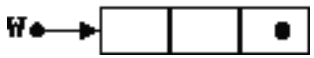
$Q(x)= 12x^3 - x^4 + 2x^5 + 24x^6$

$W(x)= P(x) + Q(x)= 2 + 5x^2 - 2x^4 + 2x^5 + 23x^6$

- Αν παραστήσουμε τα πολυώνυμα P(x) και Q(x) σαν συνδεδεμένες λίστες με κόμβους κεφαλή



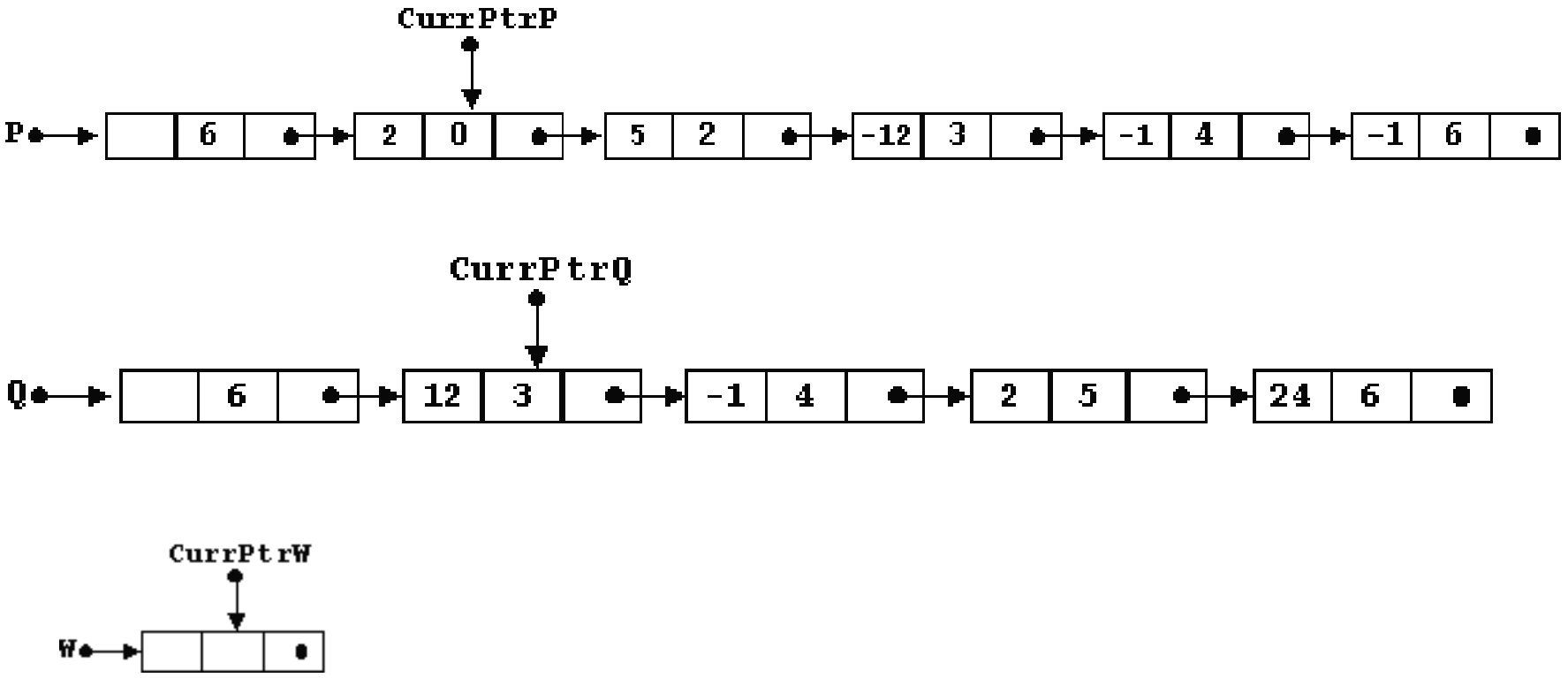
- τότε θα παραστήσουμε και το W(x) σαν συνδεδεμένη λίστα που αρχικά έχει μόνο τον κόμβο κεφαλή χωρίς τιμές:



Παράδειγμα: πρόσθεση πολυωνύμων -3-

- Χρειαζόμαστε τρεις δείκτες, CurrPtrP, CurrPtrQ και CurrPtrW:
- οι CurrPtrP και CurrPtrQ θα δείχνουν στον κόμβο των λιστών P και Q αντίστοιχα που επεξεργαζόμαστε την τρέχουσα στιγμή, ενώ
- ο CurrPtrW θα δείχνει στον τελευταίο κόμβο που προστέθηκε στη λίστα W.
- Αρχικά οι CurrPtrP και CurrPtrQ θα δείχνουν στους κόμβους P->next και Q->next και ο CurrPtrW θα δείχνει εκεί που δείχνει και ο W:

Παράδειγμα: πρόσθεση πολυωνύμων -3-



Αλγόριθμος πρόσθεσης πολυωνύμων -1-

- Κάθε φορά, συγκρίνουμε τους εκθέτες που είναι αποθηκευμένοι στους κόμβους στους οποίους δείχνουν οι CurrPtrP και CurrPtrQ.
- Αν οι εκθέτες είναι ίδιοι, τότε προσθέτουμε τους αντίστοιχους συντελεστές. Εδώ διακρίνουμε δύο περιπτώσεις:

  α) αν το αποτέλεσμα δεν είναι μηδέν, τότε προσθέτουμε έναν καινούργιο κόμβο στη λίστα W και αποθηκεύουμε το άθροισμα των συντελεστών στο τμήμα Coef του και τον κοινό εκθέτη στο τμήμα Exp.

  Οι τρεις δείκτες CurrPtrP, CurrPtrQ και CurrPtrW προχωρούν στους επόμενους κόμβους αντίστοιχα.

  β) αν το άθροισμα των συντελεστών είναι μηδέν, τότε δεν προσθέτουμε νέο κόμβο στη λίστα W, αλλά απλώς αυξάνουμε τους CurrPtrP και CurrPtrQ.

Αλγόριθμος πρόσθεσης πολυωνύμων -2-

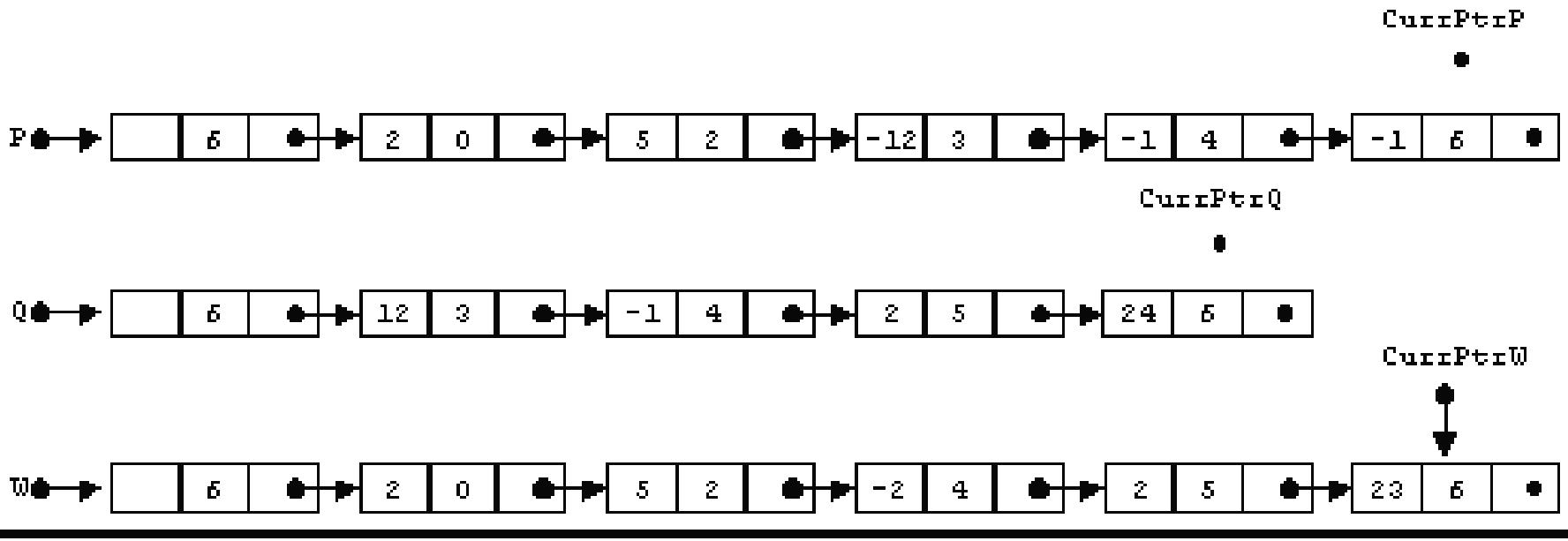
- Αν οι εκθέτες είναι διαφορετικοί, τότε προστίθεται ένας νέος κόμβος στην λίστα W, με τον μικρότερο από τους δύο εκθέτες στο τμήμα Exp και τον αντίστοιχο συντελεστή στο τμήμα Coef.
- Ο δείκτης που έδειχνε στον μικρότερο εκθέτη και ο CurrPtrW προχωρούν στους επόμενους κόμβους.
- Η διαδικασία συνεχίζεται μέχρι να φτάσουμε στο τέλος της λίστας P ή Q, δηλαδή μέχρι ένας από τους CurrPtrP και CurrPtrQ να πάρει τιμή nil.

Αλγόριθμος πρόσθεσης πολυωνύμων -3-

- Αν δεν φτάσουμε συγχρόνως στο τέλος και της άλλης λίστας, τότε αντιγράφουμε τους υπόλοιπους κόμβους και τους προσθέτουμε στη λίστα W και θέτουμε το τμήμα δεσμού του τελευταίου κόμβου ίσο με nil.
- Πιο συγκεκριμένα, η διαδικασία της πρόσθεσης των πολυωνύμων P(x) και Q(x) φαίνεται παρακάτω:

Εφαρμογή του αλγορίθμου

Οι υπολογισμοί έχουν τελειώσει. Η λίστα W περιλαμβάνει το πολυώνυμο W(x), που είναι το άθροισμα των πολυωνύμων P(x) και Q(x) και έχει βαθμό 6, όπως φαίνεται και τον κάμβο κεφαλή του.



Διαδικασία πρόσθεσης συνδεδεμένων πολυωνύμων -1-

```
void LinkedPolyAdd(PolyPointer P, PolyPointer
   Q, PolyPointer *W)
/*Δέχεται:  Δύο πολυώνυμα, P και Q.
  Λειτουργία: Υπολογίζει το άθροισμα W = P+Q.
  Επιστρέφει: Το πολυώνυμο W.
  Σημείωση: Τα πολυώνυμα παριστάνονται ως
   συνδεδεμένες λίστες με κόμβο κεφαλή.*/
```

Διαδικασία πρόσθεσης συνδεδεμένων πολυωνύμων -2-

```
{
    PolyPointer ptrP,ptrQ,ptrW,TempPtr;
    CoefficientType sum;

    ptrP = P->next;
    ptrQ = Q->next;
    *W = (PolyPointer)malloc(sizeof(struct
   PolyNode));
    ptrW = *W;
```

Διαδικασία πρόσθεσης συνδεδεμένων πολυωνύμων -3-

```
while (ptrP != NULL && ptrQ != NULL) {
    if (ptrP->Expo < ptrQ->Expo) {
      //αντιγραφή του όρου από το P
       Attach(ptrP->Coef, ptrP->Expo,&ptrW);
       ptrP = ptrP->next;
    }
    else
```

Διαδικασία πρόσθεσης συνδεδεμένων πολυωνύμων -4-

```
if (ptrP->Expo > ptrQ->Expo)   //αντιγραφή του όρου από το Q
        {
                Attach(ptrQ->Coef, ptrQ->Expo,&ptrW);
                ptrQ = ptrQ->next;
         }
         else {                          //ίδιοι εκθέτες
                sum = ptrP->Coef + ptrQ->Coef;
                if (sum != 0)
                        Attach(sum,ptrP->Expo,&ptrW);
                        ptrP = ptrP->next;
            ptrQ = ptrQ->next;
        }
   }
```

Διαδικασία πρόσθεσης συνδεδεμένων πολυωνύμων -5-

```
// αντιγραφή των υπολοίπων όρων του P ή του Q στο W
    if (ptrP != NULL)   TempPtr = ptrP;
    else TempPtr = ptrQ;
    while (TempPtr != NULL)
    {
        Attach(TempPtr-> Coef, TempPtr-> Expo,
                &ptrW);
        TempPtr = TempPtr->next;
    }
    ptrW->next = NULL;
}
```

Η διαδικασία Attach -1-

void Attach(CoefficientType Co,int Ex,
PolyPointer *Last

/*Δέχεται: ΄Έναν συντελεστή Coef, έναν εκθέτη Exp και έναν δείκτη Last.

Λειτουργία: Δημιουργεί έναν κόμβο που περιέχει τους Coef και Exp, τον συνδέει με τον κόμβο στον οποίο δείχνει ο Last και κάνει τον Last να δείχνει σ' αυτόν τον νέο κόμβο.

Επιστρέφει: Τον τροποποιημένο δείκτη Last.*/

Η διαδικασία Attach -2-

```
{
  PolyPointer TempPtr;
  TempPtr= (PolyPointer)malloc(sizeof(struct
   PolyNode));
  TempPtr->Coef = Co;
  TempPtr->Expo = Ex;
  TempPtr->next = NULL;
  (*Last)->next = TempPtr;
  *Last = TempPtr;
}
```

Χρήση πίνακα

- Αν χρησιμοποιηθεί πίνακας ως αποθηκευτική δομή των πολυωνύμων, όπως περιγράφτηκε πιο πάνω, τότε η διαδικασία της πρόσθεσης είναι αρκετά πιο απλή.
- Το μόνο που χρειάζεται είναι ο παρακάτω βρόχος for:

```
for (i = 0; i <= MaxDegree; i++)
        W[i] = P[i]+Q[i];
```

όπου MaxDegree είναι ο μέγιστος βαθμός των πολυωνύμων P και Q.

Τέλος Ενότητας

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης