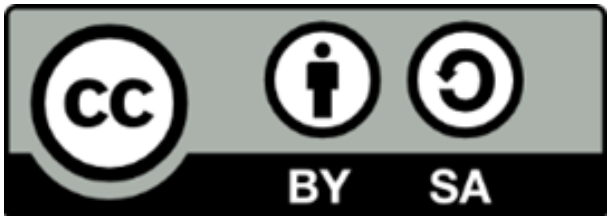
Δομές Δεδομένων

Ενότητα 4: Ο ΑΤΔ Λίστα & Υλοποίηση Λίστας με σειριακή αποθήκευση- Ο ΑΤΔ Συνδεδεμένη Λίστα- Υλοποίηση ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

Καθηγήτρια Μαρία Σατρατζέμη
Τμήμα Εφαρμοσμένης Πληροφορικής

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

- Κατανόηση του ΑΤΔ Λίστα.
- Σειριακή αποθήκευση λίστας
- Υλοποίηση των βασικών λειτουργιών
- Να μελετηθεί ο ΑΤΔ λίστα, η υλοποίηση της με πίνακα και δομή καθώς και η υλοποίηση των πράξεων που περιέχει η λίστα.
- Να μελετηθεί η Συνδεδεμένη Λίστα και να δοθεί η υλοποίηση της με πίνακα και δομή καθώς και η υλοποίηση των πράξεων που περιέχει η Συνδεδεμένη λίστα.

Περιεχόμενα ενότητας[1]

- Εισαγωγή στον ΑΤΔ Λίστα
- Διαφορά λίστας, στοίβας, ουράς
- Ορισμός ΑΤΔ λίστα
- Αποθηκευτική δομή ΑΤΔ λίστα
- Παράδειγμα Εισαγωγής, διαγραφής
- Ένα πακέτο για τον ΤΔ λίστα (σειριακή αποθήκευση)

Περιεχόμενα ενότητας[2]

- Εισαγωγή ΑΤΔ Συνδεδεμένη Λίστα
- Βασικές λειτουργίες συνδεδεμένων λιστών
- Δημιουργία κενής Σ.Λ.
- Έλεγχος κενής ΣΛ
- Εισαγωγή
- Διαγραφή
- Διάσχιση

Περιεχόμενα ενότητας[3]

- Εισαγωγή Συνδεδεμένη Λίστα με πίνακα
- Δηλώσεις
- Παράδειγμα
- Δεξαμενή διαθέσιμων κόμβων
- Αρχικοποίηση δεξαμενής
- Πακέτο για τη Συνδεδεμένη λίστα με πίνακα

Εισαγωγή

Ως δομή δεδομένων, μια **λίστα** (**list**) είναι μια πεπερασμένη αλληλουχία στοιχείων.

Αν και οι **βασικές λειτουργίες** που συνδέονται με τις λίστες διαφέρουν ανάλογα με την εφαρμογή, συνήθως περιλαμβάνουν τα ακόλουθα:

- Δημιουργία κενής λίστας
- Έλεγχος αν μια λίστα είναι κενή
- Διάσχιση της λίστας ή τμήματός της για προσπέλαση και επεξεργασία των στοιχείων με τη σειρά
- Εισαγωγή ενός νέου στοιχείου στη λίστα
- Διαγραφή κάποιου στοιχείου από τη λίστα.

Ο ΑΤΔ Λίστα & Υλοποίηση Λίστας με σειριακή αποθήκευση

Διαφορά λίστας, στοίβας και ουράς

Βασική διαφορά της λίστας από την στοίβα και την ουρά:

- η λίστα είναι προσπελάσιμη σε οποιαδήποτε θέση της,
- η στοίβα είναι προσπελάσιμη μόνο στο ένα άκρο της,
- η ουρά είναι προσπελάσιμη στα δύο άκρα της.

Έτσι, λοιπόν, η εισαγωγή ενός στοιχείου σε μια λίστα μπορεί να γίνει σε οποιαδήποτε θέση της (ανάλογα με το πρόβλημα) και η διαγραφή ενός στοιχείου μπορεί να γίνει από οποιαδήποτε θέση της.

Ορισμός του ΑΤΔ Λίστα -1-

Συλλογή στοιχείων δεδομένων:
Μια ακολουθία στοιχείων δεδομένων σε γραμμική διάταξη.

Βασικές λειτουργίες:

Δημιουργία κενής λίστας (**CreateList**)

Λειτουργία: Δημιουργεί μια κενή λίστα.

Επιστρέφει: Μια κενή λίστα.

Ορισμός του ΑΤΔ Λίστα -2-

Έλεγχος κενής λίστας(EmptyList)

Δέχεται: Μια λίστα.

Λειτουργία: Ελέγχει αν η λίστα είναι κενή.

Επιστρέφει: TRUE, αν η λίστα είναι κενή, FALSE διαφορετικά.

Ορισμός του ΑΤΔ Λίστα -3-

Διάσχιση (**Traverse**)

Δέχεται: Μια λίστα.

Λειτουργία: Διασχίζει τη λίστα (ή τμήμα της) για προσπέλαση και επεξεργασία των στοιχείων με τη σειρά.

Επιστρέφει: Εξαρτάται από το είδος της επεξεργασίας.

Ορισμός του ΑΤΔ Λίστα -4-

Διάσχιση (**Traverse**)

Δέχεται: Μια λίστα.

Λειτουργία: Διασχίζει τη λίστα (ή τμήμα της) για προσπέλαση και επεξεργασία των στοιχείων με τη σειρά.

Επιστρέφει: Εξαρτάται από το είδος της επεξεργασίας.

Ορισμός του ΑΤΔ Λίστα -5-

Διαγραφή στοιχείου (**Delete**)

Δέχεται: Μια λίστα και μια θέση μέσα στη λίστα.

Λειτουργία: Διαγράφει το στοιχείο από την συγκεκριμένη θέση της λίστας.

Επιστρέφει: Την τροποποιημένη λίστα.

Αποθηκευτική Δομή -1-

Καθότι οι λίστες, όμοια με τις στοίβες και τις ουρές, είναι ακολουθίες στοιχείων δεδομένων, θα ήταν φυσικό να χρησιμοποιήσουμε και πάλι έναν πίνακα για τη βασική αποθηκευτική δομή.

Με την μέθοδο αυτή τα διαδοχικά στοιχεία της λίστας αποθηκεύονται σε μία σειρά διαδοχικών θέσεων πίνακα, με το πρώτο στοιχείο της λίστας στην θέση 1 του πίνακα, το δεύτερο στην θέση 2, κ.ο.κ.

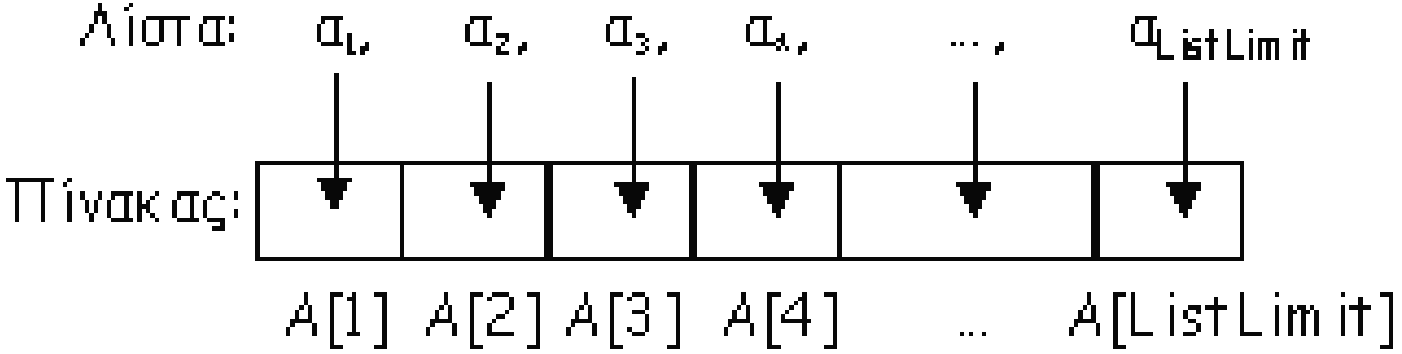
Αποθηκευτική Δομή -2-

Καθότι οι λίστες, όμοια με τις στοίβες και τις ουρές, είναι ακολουθίες στοιχείων δεδομένων, θα ήταν φυσικό να χρησιμοποιήσουμε και πάλι έναν πίνακα για τη βασική αποθηκευτική δομή.

Με την μέθοδο αυτή τα διαδοχικά στοιχεία της λίστας αποθηκεύονται σε μία σειρά διαδοχικών θέσεων πίνακα, με το πρώτο στοιχείο της λίστας στην θέση 1 του πίνακα, το δεύτερο στην θέση 2, κ.ο.κ.

Αποθηκευτική Δομή -3-

Θα αναφερόμαστε σε αυτήν την υλοποίηση λίστας ως την **υλοποίηση λίστας με σειριακή αποθήκευση** (**sequential storage implemention of lists**).



Αποθηκευτική Δομή -4-

Μια λίστα και τα στοιχεία της μπορούν να αποθηκευτούν με τη χρήση μιας εγγραφής, όπως φαίνεται παρακάτω:

```
#define ListLimit 50

typedef struct {
        int Size;
        ListElementType Element[ListLimit];
} ListType;
```

Κενή λίστα

Οι τρεις πρώτες βασικές λειτουργίες της λίστας είναι εύκολο να υλοποιηθούν.

Αν η List είναι τύπου ListType τότε:

- Δημιουργία κενής λίστας (CreateList): Μια κενή λίστα δημιουργείται θέτοντας **List.Size = 0**.
- Έλεγχος κενής λίστας (EmptyList): Αν θεωρήσουμε ότι υπάρχει μια boolean συνάρτηση EmptyList, τότε μπορεί να πάρει τιμή TRUE ή FALSE αν η συνθήκη **EmptyList == (List.Size=0)** είναι αληθής ή ψευδής αντίστοιχα

Διάσχιση λίστας

- Traverse: Η διάσχιση της λίστας για π.χ. εμφάνιση των στοιχείων της μπορεί να γίνει με τον ακόλουθο βρόχο:

Για i από 0 **μέχρι**
List.Size - 1
Γράψε List.Element[i]
Τέλος_επανάληψης

Παράδειγμα

Για να δούμε πώς λειτουργούν οι διαδικασίες εισαγωγής και διαγραφής στοιχείων από την ή στην λίστα, ας θεωρήσουμε μια λίστα ταξινομημένων ακεραίων αριθμών με αύξουσα διάταξη όπως η ακόλουθη:

4, 12, 37, 40, 49, 63, 71

Η λίστα αυτή μπορεί να αποθηκευτεί με την μορφή πίνακα ως εξής:

List

θέση	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**...**
αριθμός	4	12	37	40	49	63	71	...

Παράδειγμα: Εισαγωγή

- Έστω ότι θέλουμε να προσθέσουμε ένα νέο αριθμό στην λίστα, π.χ. τον 28.
- Επειδή θέλουμε τα στοιχεία της λίστας να είναι ταξινομημένα κατά αύξουσα σειρά, θα πρέπει να μετατοπίσουμε κατά μία θέση δεξιά τα στοιχεία που βρίσκονται δεξιά του 12, ώστε να δημιουργηθεί μια ελεύθερη θέση μεταξύ του 12 και του 37 και να τοποθετηθεί εκεί ο αριθμός 28, όπως φαίνεται παρακάτω:

List

θέση	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
αριθμός	4	12	28	37	40	49	63	71	...

Παράδειγμα: λίστα γεμάτη;

- Πριν προχωρήσουμε στην εισαγωγή του νέου στοιχείου στη λίστα, θα πρέπει να εξετάσουμε αν υπάρχει χώρος για αυτό το νέο στοιχείο, δηλαδή αν η λίστα δεν είναι γεμάτη.
- Ενώ, λοιπόν, θεωρητικά μια λίστα είναι απεριόριστη και μπορούμε να εισάγουμε όσα στοιχεία θέλουμε σ' αυτήν, εδώ περιοριζόμαστε από το μέγεθος του πίνακα με τον οποίο υλοποιούμε την λίστα.

Συνάρτηση για λίστα γεμάτη;

- Επομένως, είναι απαραίτητο να συμπεριληφθεί και μια **συνάρτηση FullList** στο πακέτο για τον ΑΤΔ Λίστα, η οποία να εξετάζει αν προκλήθηκε κάποιο σφάλμα στην προσπάθεια εισαγωγής ενός νέου στοιχείου στη λίστα.

Παράδειγμα: Διαγραφή

- Έστω τώρα ότι θέλουμε να διαγράψουμε τον αριθμό 12 από τη λίστα που είχαμε προηγουμένως.
- Η διαγραφή αυτή συνεπάγεται και μετατόπιση κατά μία θέση προς τα αριστερά των αριθμών που βρίσκονταν δεξιά του 12.

List

θέση	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**...**
αριθμός	4	28	37	40	49	63	71	...

Ένα πακέτο για τη Λίστα με πίνακα -1-

```
// Filename ListADT.h
#define ListLimit 50
typedef int ListElementType;
typedef struct {
        int Size;
         ListElementType Element[ListLimit];
} ListType;
typedef enum {
        FALSE, TRUE
} boolean;
```

Ένα πακέτο για τη Λίστα με πίνακα -2-

```
void CreateList(ListType *List);
boolean EmptyList(ListType List);
boolean FullList(ListType List);
void Insert(ListType *List, ListElementType Item, int Pos);
void Delete(ListType *List, int Pos);
void TraverseList(ListType List);
```

Ένα πακέτο για τη Λίστα με πίνακα -3-

```
// Filename ListADT.c

void CreateList(ListType *List)
/*Λειτουργία:  Δημιουργεί μια κενή λίστα.
     Επιστρέφει:  Μια κενή λίστα*/
   {
   (*List).Size = 0;
}
```

Ένα πακέτο για τη Λίστα με πίνακα -4-

```
boolean EmptyList(ListType List)
/*Δέχεται:  Μια λίστα List.
  Λειτουργία: Ελέγχει αν η λίστα List είναι κενή.
  Επιστρέφει: TRUE αν η λίστα List είναι άδεια,
   FALSE διαφορετικά.*/
{
    return (List.Size == 0);
}
```

Ένα πακέτο για τη Λίστα με πίνακα -5-

```
boolean FullList(ListType List)
/*Δέχεται:  Μια λίστα List.
  Λειτουργία: Ελέγχει αν η λίστα List είναι
   γεμάτη.
  Επιστρέφει: TRUE αν η λίστα List είναι
   γεμάτη, FALSE διαφορετικά.*/
{
    return (List.Size == (ListLimit));
}
```

Ένα πακέτο για τη Λίστα με πίνακα -6-

void Insert(ListType *List,
ListElementType Item, **int** Pos)

/*Δέχεται: Μια λίστα *List*, ένα στοιχείο *Item* και μια θέση *Pos* μέσα στη λίστα.

Λειτουργία: Εισάγει το στοιχείο *Item* στη λίστα *List* μετά από το στοιχείο που βρίσκεται στη θέση *Pos* (αν η λίστα είναι κενή τότε *Pos*=0).

Επιστρέφει: Την τροποποιημένη λίστα.

Έξοδος: Μήνυμα λάθους στην περίπτωση που η εισαγωγή αποτύχει.*/.

΄Ενα πακέτο για τη Λίστα με πίνακα -7-

```
{
        int i;
        if (FullList(*List)) printf("Full list...\n");
        else {
          for (i= (*List).Size-1; i>=Pos+1;i- -)
                        (*List).Element[i+1] = (*List).Element[i];
          (*List).Element[Pos+1] = Item;
          (*List).Size++;
        }
}
```

Ένα πακέτο για τη Λίστα με πίνακα -8-

void Delete(ListType *List, **int** Pos)

/*Δέχεται: Μια λίστα *List* και μια θέση *Pos* μέσα στη λίστα.

Λειτουργία: Διαγράφει το στοιχείο από τη θέση *Pos* της λίστας *List*.

Επιστρέφει: Την τροποποιημένη λίστα.

Έξοδος: Μήνυμα λάθους στην περίπτωση που η διαγραφή αποτύχει.*/

Ένα πακέτο για τη Λίστα με πίνακα -9-

```
{
    int i;
    if (EmptyList(*List)) printf("Empty list...\n");
    else {
        for (i = Pos; i< (*List).Size-1; i++)
                (*List).Element[i] = (*List).Element[i+1];
        (*List).Size--;
    }
}
```

Αποδοτικότητα της διαδικασίας Insert -1-

- Η αποδοτικότητα της διαδικασίας Insert εξαρτάται από το πλήθος των στοιχείων του πίνακα που πρέπει να μετακινηθούν προκειμένου να δημιουργηθεί κενή θέση για το νέο στοιχείο.

Αποδοτικότητα της διαδικασίας Insert -2-

- Στην **καλύτερη περίπτωση** το νέο στοιχείο πρέπει να εισαχθεί στο τέλος της λίστας, οπότε δεν χρειάζεται να μετακινηθεί κανένα από τα στοιχεία του πίνακα.
- Στην **χειρότερη περίπτωση** το νέο στοιχείο πρέπει να εισαχθεί στην αρχή της λίστας, πράγμα το οποίο συνεπάγεται μετατόπιση όλων των στοιχείων της λίστας.
- Επομένως, αν το μέγεθος της λίστας είναι *n* τότε **κατά μέσο όρο πρέπει να μετακινηθούν n/2 στοιχεία** του πίνακα για την εισαγωγή ενός νέου στοιχείου.

Αποδοτικότητα της διαδικασίας Insert -3-

- Βέβαια, όλα τα παραπάνω ισχύουν εφόσον μας ενδιαφέρει η σειρά των στοιχείων στη λίστα. Αν η σειρά δεν μας ενδιαφέρει, τότε η εισαγωγή νέων στοιχείων μπορεί να γίνει σε οποιαδήποτε θέση της λίστας και μάλιστα βολεύει να γίνεται στο τέλος της. Για μια τέτοια λίστα η εισαγωγή στοιχείου είναι ίδια με την ώθηση ενός στοιχείου σε μια στοίβα.

Αποδοτικότητα της διαδικασίας Delete

- Η αποδοτικότητα της διαδικασίας Delete εξαρτάται και αυτή από το πλήθος των στοιχείων του πίνακα που πρέπει να μετακινηθούν μετά τη διαγραφή του στοιχείου:
  - Στη **καλύτερη περίπτωση** το στοιχείο διαγράφεται από το τέλος της λίστας, οπότε δεν χρειάζεται να μετακινηθεί κανένα από τα στοιχεία του πίνακα.
  - Στην **χειρότερη περίπτωση** το στοιχείο διαγράφεται από την αρχή της λίστας, πράγμα το οποίο συνεπάγεται μετατόπιση όλων των στοιχείων της λίστας.
  - Επομένως, και πάλι ο **μέσος όρος μετακινήσεων** που πρέπει να γίνουν είναι **n/2.**

Ο ΑΤΔ Συνδεδεμένη Λίστα

Εισαγωγή -1-

- Μια λίστα είναι μια ακολουθία στοιχείων δεδομένων, πράγμα το οποίο σημαίνει ότι υπάρχει μία διάταξη όσον αφορά τα στοιχεία της λίστας: κάποιο στοιχείο είναι πρώτο, κάποιο δεύτερο, κ.ο.κ.

- Γι' αυτό, οποιαδήποτε υλοποίηση αυτής της δομής δεδομένων πρέπει να έχει ενσωματωμένη μια μέθοδο που να καθορίζει αυτήν την διάταξη.

Εισαγωγή -2-

- Στην υλοποίηση με σειριακή αποθήκευση, η διάταξη των στοιχείων της λίστας δινόταν *σιωπηρά* από την φυσική διάταξη των στοιχείων του πίνακα, αφού το πρώτο στοιχείο ήταν αποθηκευμένο στην πρώτη θέση του πίνακα, το δεύτερο στοιχείο στην δεύτερη θέση, κ.ο.κ.

- Αυτός ο σιωπηρός προσδιορισμός της διάταξης των στοιχείων της λίστας ήταν που έκανε απαραίτητη την μετατόπισή τους στον πίνακα σε κάθε εισαγωγή ή διαγραφή στοιχείου.

Εισαγωγή -3-

- Στην ενότητα αυτή, θεωρούμε μια εναλλακτική υλοποίηση λίστας, στην οποία η διάταξη των στοιχείων δίνεται *ρητά*.

Εισαγωγή -4-

- Στην ενότητα αυτή, θεωρούμε μια εναλλακτική υλοποίηση λίστας, στην οποία η διάταξη των στοιχείων δίνεται *ρητά*.
- Σε οποιαδήποτε δομή, που χρησιμοποιείται για την αποθήκευση στοιχείων λίστας, πρέπει να υπάρχει η δυνατότητα εκτέλεσης τουλάχιστον των παρακάτω **λειτουργιών**, αν πρέπει να διατηρηθεί η διάταξη των στοιχείων της λίστας:
  - Εντοπισμός του πρώτου στοιχείου
  - Δοσμένης της θέσης οποιουδήποτε στοιχείου, προσδιορισμός της θέσης του στοιχείου που ακολουθεί

Εισαγωγή -5-

- Στην υλοποίηση με σειριακή αποθήκευση η διάταξη δίνεται σιωπηρά από τους δείκτες του πίνακα, δηλαδή το πρώτο στοιχείο της λίστας αποθηκεύεται στην θέση 1 του πίνακα, το δεύτερο στοιχείο στην θέση 2, κλπ, και το επόμενο του στοιχείου της θέσης i βρίσκεται στην θέση i+1 του πίνακα.

Εισαγωγή -6-

Μια δομή αποθήκευσης στοιχείων λίστας στην οποία η διάταξη δίνεται ρητά ονομάζεται **συνδεδεμένη λίστα** (**linked lists**).

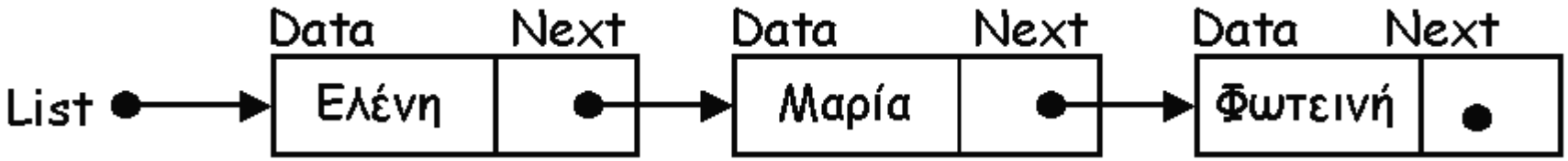
Μια συνδεδεμένη λίστα είναι μια συλλογή στοιχείων, που ονομάζονται **κόμβοι** (**nodes**)**,** και σε κάθε κόμβο αποθηκεύονται δύο είδη πληροφορίας:

- Ένα στοιχείο της λίστας
- Ένας **δεσμός** (**link**) ή **δείκτης** (**pointer**) που δείχνει ρητά τη θέση του κόμβου που περιέχει το επόμενο στοιχείο της λίστας

Επίσης, πρέπει να διατηρείται πρόσβαση στον κόμβο όπου είναι αποθηκευμένο το πρώτο στοιχείο της λίστας.

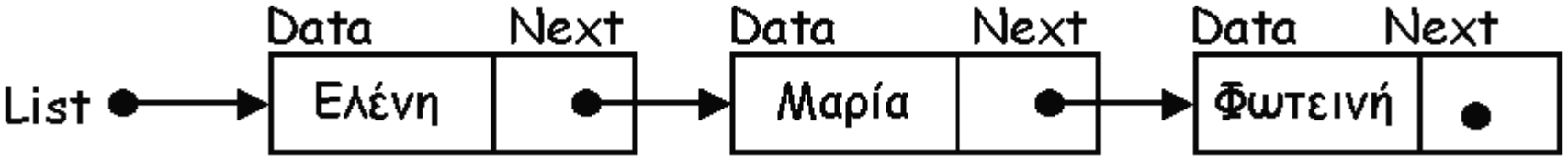
Παράδειγμα -1-

Έστω ότι έχουμε μια συνδεδεμένη λίστα όπου αποθηκεύουμε τα ονόματα *Ελένη*, *Μαρία* και *Φωτεινή*, όπως φαίνεται στο παρακάτω σχήμα.



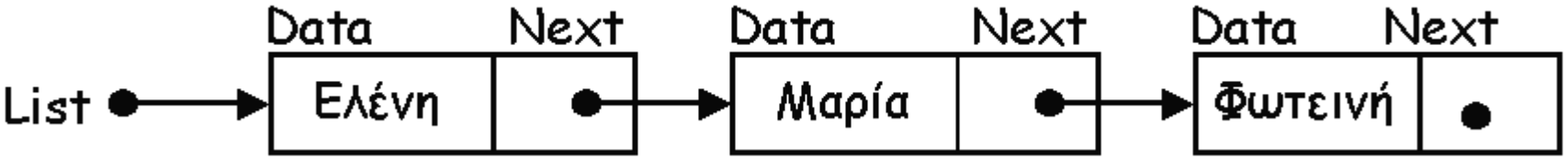
- Στο σχήμα αυτό, τα βέλη παριστάνουν τους δεσμούς και ο δείκτης **List** δείχνει στον πρώτο κόμβο της λίστας.

Παράδειγμα -2-



- Στο σχήμα αυτό, τα βέλη παριστάνουν τους δεσμούς και ο δείκτης **List** δείχνει στον πρώτο κόμβο της λίστας.
- Στο τμήμα δεδομένων (**Data**) του κάθε κόμβου αποθηκεύεται ένα από τα ονόματα της λίστας και η κουκίδα στον τελευταίο κόμβο που δεν έχει βέλος να ξεκινάει από αυτήν αντιπροσωπεύει έναν **μηδενικό δείκτη** (**nil pointer**) και δηλώνει ότι δεν υπάρχει επόμενο γι' αυτό το στοιχείο.

Παράδειγμα -3-



- Αν με **p** συμβολίσουμε το δείκτη προς κάποιο κόμβο αυτής της λίστας, τότε θα δηλώνουμε το τμήμα δεδομένου του κόμβου ως **Data(p)** και το τμήμα δεσμού ως **Next(p)**.
- Επίσης, θα θεωρούμε ότι **NilValue** είναι μια ειδική τιμή που μπορεί να δοθεί στο p για να δείξει ότι είναι μηδενικός δείκτης, δηλαδή δε δείχνει σε κανένα κόμβο.

Βασικές λειτουργίες -1-

Δημιουργία κενής λίστας – CreateList

Για να δημιουργήσουμε μια κενή λίστα αρκεί να θέσουμε την τιμή NilValue στον δείκτη List για να δείξουμε ότι δεν δείχνει σε κανένα κόμβο,δηλαδή:

List ●

Έλεγχος κενής λίστας - EmptyList

Για να καθορίσουμε αν η λίστα είναι κενή μπορούμε απλά να εξετάσουμε αν ο δείκτης List έχει την τιμή NilValue.

Βασικές λειτουργίες -2-

Διάσχιση λίστας – Traverse

Έστω ότι θέλουμε να διασχίσουμε μια λίστα όπως η παραπάνω λίστα ονομάτων.

- Χρησιμοποιούμε έναν **βοηθητικό δείκτη CurrP**, ο οποίος δείχνει αρχικά στον πρώτο κόμβο, και επεξεργαζόμαστε το στοιχείο *Ελένη* που είναι αποθηκευμένο στον κόμβο αυτό.
- Για να μετακινηθούμε στον επόμενο κόμβο και να επεξεργαστούμε το στοιχείο *Μαρία*, ακολουθούμε τον δεσμό του τρέχοντος κόμβου (Next(CurrP)) θέτοντας CurrP=Next(CurrP).
- Ομοίως, για να επεξεργαστούμε το επόμενο στοιχείο, *Φωτεινή*, θέτουμε πάλι CurrP=Next(CurrP), οπότε ο CurrP δείχνει τώρα στον τρίτο κόμβο της λίστας.
- Το στοιχείο *Φωτεινή* είναι το τελευταίο της λίστας, οπότε αν θέσουμε πάλι CurrP=Next(CurrP), ο CurrP γίνεται μηδενικός και καταλαβαίνουμε ότι φτάσαμε στο τέλος της λίστας.

Βασικές λειτουργίες -3-

Εισαγωγή στοιχείου – Insert

Για να εισαγάγουμε ένα νέο στοιχείο στη λίστα, χρειάζεται πρώτα να αποκτήσουμε ένα νέο κόμβο, ώστε να αποθηκεύσουμε την τιμή του στοιχείου στο τμήμα δεδομένου του κόμβου αυτού.

Υποθέτουμε ότι υπάρχει μια *δεξαμενή* (*storage pool*) διαθέσιμων κόμβων καθώς και ένας μηχανισμός απόκτησης τέτοιων κόμβων από τη δεξαμενή.

Βασικές λειτουργίες -4-

Υποθέτουμε, δηλαδή, ότι υπάρχει μια **διαδικασία GetNode** η οποία, αν κληθεί με μια πρόταση της μορφής GetNode(TempP), θα επιστρέψει έναν προσωρινό δείκτη TempP προς έναν διαθέσιμο κόμβο.

Το επόμενο βήμα είναι να συνδεθεί αυτός ο κόμβος με την υπάρχουσα λίστα.

Διακρίνουμε δύο περιπτώσεις:

α) η εισαγωγή να γίνει στην αρχή της λίστας

β) η εισαγωγή να γίνει μετά από κάποιο στοιχείο της λίστας

Βασικές λειτουργίες -5-

Α) Εισαγωγή στοιχείου στην αρχή της λίστας

Έστω ότι θέλουμε να εισαγάγουμε το όνομα *Δήμητρα* στην αρχή της λίστας. Κατ' αρχήν

1. παίρνουμε έναν νέο κόμβο με την εντολή GetNode(TempP), στον οποίο δείχνει προσωρινά ο δείκτης TempP, και
2. αποθηκεύουμε την τιμή *Δήμητρα* στο τμήμα δεδομένου του θέτοντας Data(TempP)='Δήμητρα'. Στη συνέχεια,
3. εισάγουμε τον κόμβο αυτό στην αρχή της λίστας κάνοντας το τμήμα δεσμού του να δείχνει στον πρώτο κόμβο της λίστας, δηλαδή θέτουμε Next(TempP)=List, και, τέλος,
4. θέτουμε List=TempP ώστε ο List να δείχνει στον νέο κόμβο.

Βασικές λειτουργίες -6-

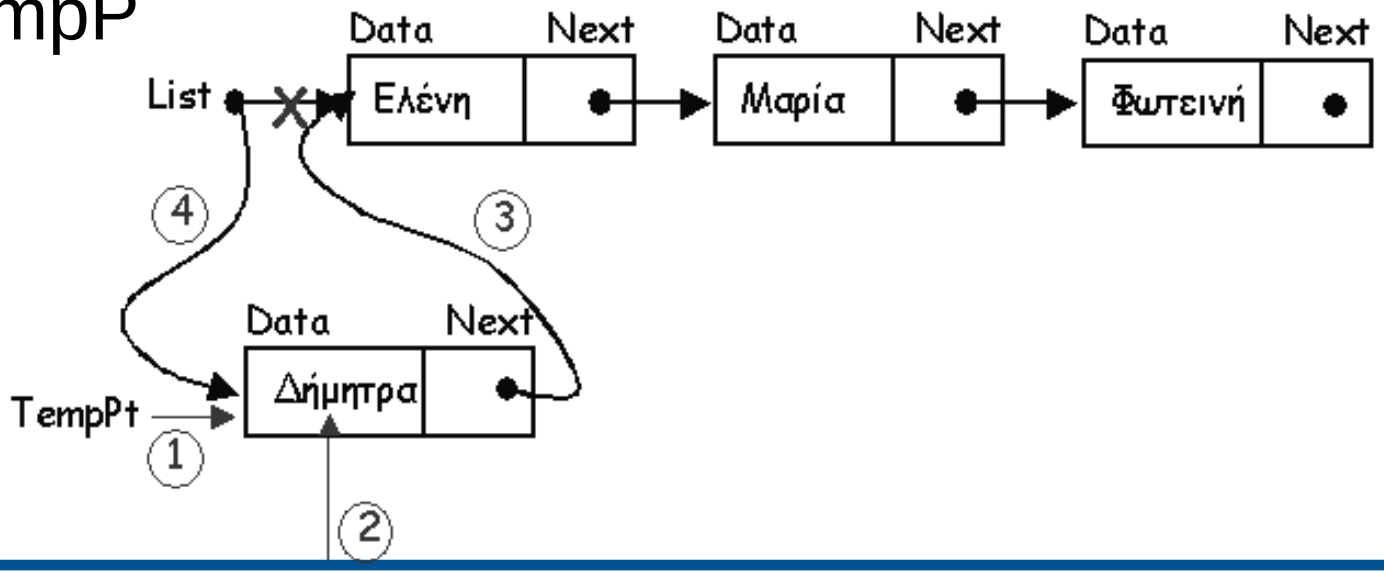
Συνολικά, δηλαδή, εκτελούμε τις ακόλουθες εντολές:

1. GetNode(TempP)
2. Data(TempP)='Δήμητρα'
3. Next(TempP)=List
4. List=TempP

Βασικές λειτουργίες -7-

Εισαγωγή στοιχείου στην αρχή της λίστας (… συνέχεια)

1. GetNode(TempP)
2. Data(TempP)='Δήμητρα'
3. Next(TempP)=List
4. List=TempP



Βασικές λειτουργίες -8-

Β) Εισαγωγή μετά από κάποιο στοιχείο της λίστας

Έστω ότι θέλουμε να εισαγάγουμε το όνομα *Βάσω* μετά από τον κόμβο που περιέχει το όνομα *Μαρία* και ότι ο δείκτης PredP δείχνει σ' αυτόν τον κόμβο:

1. Παίρνουμε πάλι έναν νέο κόμβο με την εντολή GetNode(TempP), στον οποίο δείχνει προσωρινά ο δείκτης TempP, και
2. αποθηκεύουμε την τιμή *Στέλλα* στο τμήμα δεδομένου του θέτοντας Data(TempP)='Βάσω'. Εισάγουμε τον κόμβο αυτό στη λίστα και
3. θέτουμε το τμήμα δεσμού του ίσο με Next(PredP) ώστε να δείχνει στον επόμενο του κόμβου που περιέχει το όνομα *Μαρία* με την εντολή Next(TempP)=Next(PredP) και
4. αλλάζουμε το τμήμα δεσμού τού προηγούμενου κόμβου ώστε να δείχνει στον νέο κόμβο, δηλαδή θέτουμε Next(PredP)=TempP..

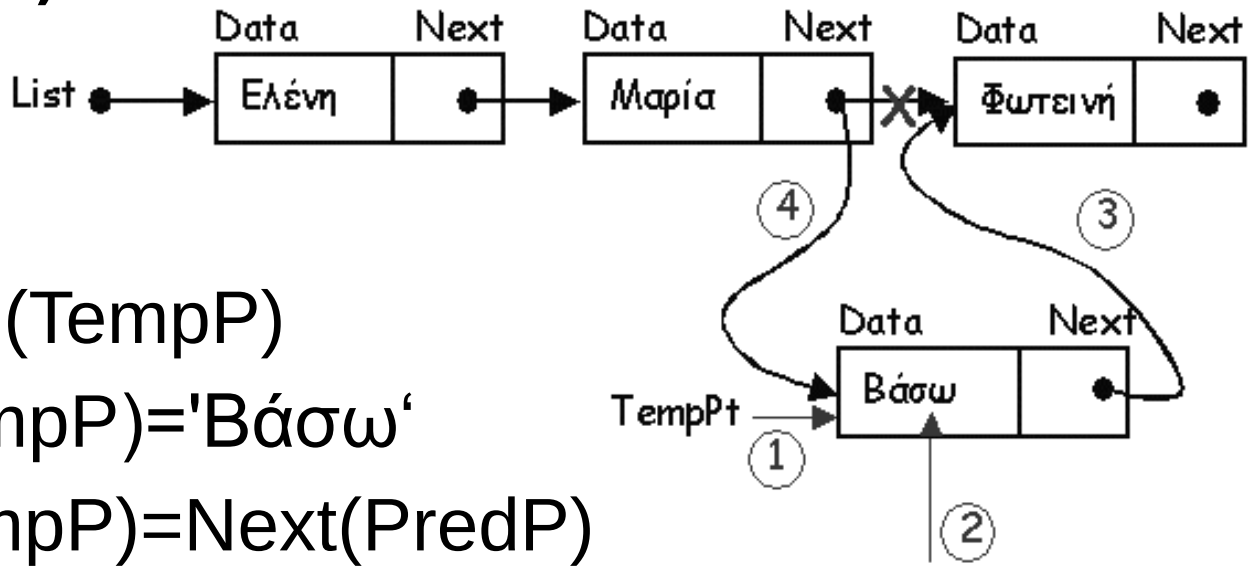
Βασικές λειτουργίες -9-

Συνολικά, δηλαδή, εκτελούμε τις ακόλουθες εντολές:

1. GetNode(TempP)
2. Data(TempP)='Βάσω‘
3. Next(TempP)=Next(PredP)
4. Next(PredP)=TempP

Βασικές λειτουργίες -10-

Εισαγωγή στοιχείου στην αρχή της λίστας (… συνέχεια)



1. GetNode(TempP)
2. Data(TempP)='Βάσω‘
3. Next(TempP)=Next(PredP)
4. Next(PredP)=TempP

Βασικές λειτουργίες -11-

Διαγραφή στοιχείου – Delete

Στην διαγραφή ενός στοιχείου από μια λίστα έχουμε πάλι δύο περιπτώσεις:

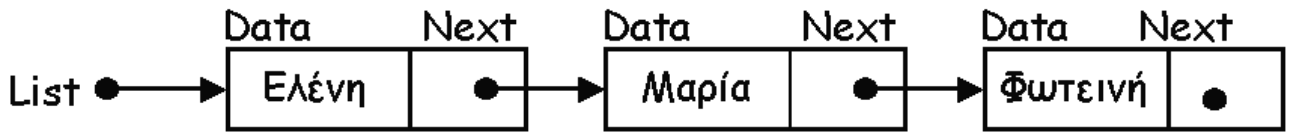
α) διαγραφή του πρώτου στοιχείου της λίστας

β) διαγραφή ενός στοιχείου από κόμβο για τον οποίο υπάρχει κάποιος προηγούμενος κόμβος

Βασικές λειτουργίες -12-

Α) Διαγραφή του πρώτου στοιχείου της λίστας

Έστω ότι θέλουμε να διαγράψουμε το στοιχείο *Ελένη* που βρίσκεται στην αρχή της λίστας του Σχήματος



Βασικές λειτουργίες -13-

Α) Διαγραφή του πρώτου στοιχείου της λίστας

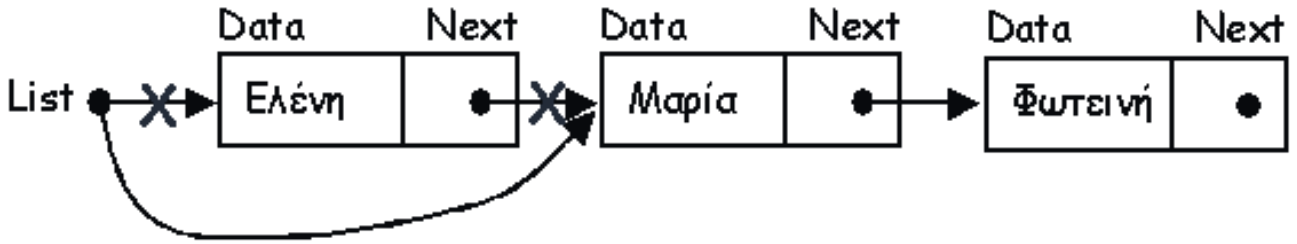
Αυτό είναι πολύ εύκολο να γίνει αν αλλάξουμε τον δείκτη List ώστε να δείχνει στον δεύτερο κόμβο της λίστας, θέτοντας TempP=List και μετά List=Next(List), και να επιστρέψουμε τον διαγραμμένο κόμβο στην δεξαμενή με τους διαθέσιμους κόμβους καλώντας μια διαδικασία ReleaseNode, δηλαδή ReleaseNode(TempP).

Βασικές λειτουργίες -14-

Α) Διαγραφή του πρώτου στοιχείου της λίστας

Συνολικά,δηλαδή, εκτελούμε τις ακόλουθες εντολές:

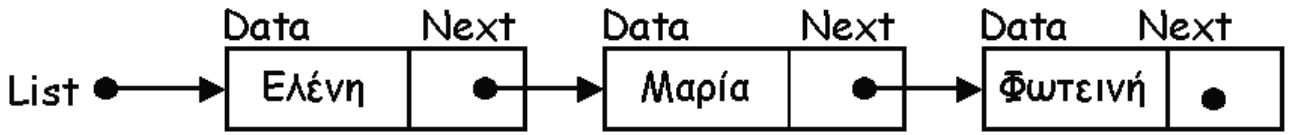
TempP=List
List=Next(List)
ReleaseNode(TempP)



Βασικές λειτουργίες -15-

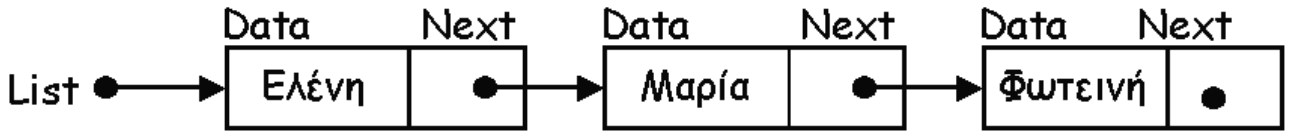
Β) Διαγραφή στοιχείου από κόμβο για τον οποίο υπάρχει κάποιος προηγούμενος κόμβος

Έστω ότι θέλουμε να διαγράψουμε τον κόμβο με το όνομα *Μαρία* της λίστας του Σχήματος



Βασικές λειτουργίες -16-

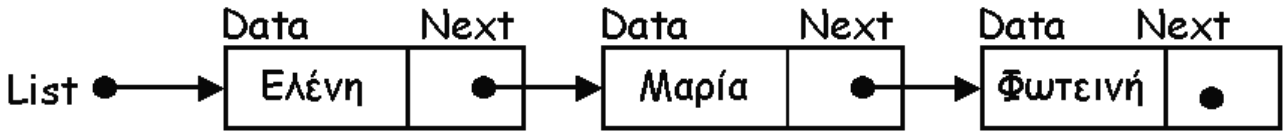
Β) Διαγραφή στοιχείου από κόμβο για τον οποίο υπάρχει κάποιος προηγούμενος κόμβος



Το μόνο που χρειάζεται να κάνουμε είναι να δείχνει ο προηγούμενος κόμβος αυτού που θέλουμε να διαγραφεί στον επόμενό του, δηλαδή TempP=Next(PredP) και μετά Next(PredP)=Next(TempP), και, τέλος, να καλέσουμε την ReleaseNode για να επιστρέψουμε τον διαγραμμένο κόμβο στη δεξαμενή των διαθέσιμων κόμβων, ReleaseNode(TempP).

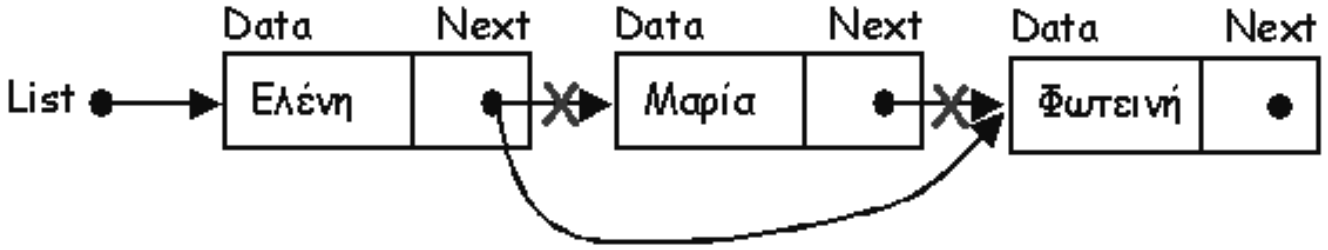
Βασικές λειτουργίες -17-

Β) Διαγραφή στοιχείου από κόμβο για τον οποίο υπάρχει κάποιος προηγούμενος κόμβος



Συνολικά, δηλαδή, εκτελούμε τις ακόλουθες εντολές:

TempP=Next(PredP)
Next(PredP)=Next(TempP)
ReleaseNode(TempP)



Πλεονέκτημα των συνδεδεμένων λιστών

Οι εισαγωγές και διαγραφές στοιχείων

- μπορούν να γίνουν σε οποιαδήποτε θέση της λίστας
- ανεξάρτητα από το πλήθος των στοιχείων της και
- χωρίς μετατοπίσεις στοιχείων..

Υλοποίηση ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

Εισαγωγή -1-

Επειδή οι περισσότερες γλώσσες προγραμματισμού δεν περιλαμβάνουν κάποιο προκαθορισμένο τύπο δεδομένων για τις συνδεδεμένες λίστες, η υλοποίησή τους μπορεί να γίνει με χρήση άλλων τύπων δεδομένων.

Εδώ θα ασχοληθούμε με την **υλοποίηση των συνδεδεμένων λιστών με χρήση πινάκων και εγγραφών**.

Εισαγωγή -2-

Οι κόμβοι μιας συνδεδεμένης λίστας περιλαμβάνουν:

- το ***τμήμα δεδομένου*** (Data), όπου αποθηκεύεται ένα στοιχείο της λίστας, και
- το ***τμήμα δεσμού*** (Link), όπου αποθηκεύεται ένας δείκτης, ο οποίος είτε δείχνει στον επόμενο κόμβο της λίστας είτε είναι μηδενικός, στην περίπτωση που είναι ο δείκτης του τελευταίου κόμβου.

Εισαγωγή -3-

Κάθε κόμβος μπορεί να παρασταθεί με μια εγγραφή και η συνδεδεμένη λίστα με έναν πίνακα τέτοιων εγγραφών.

Κάθε εγγραφή θα αποτελείται:

- από ένα **πεδίο Data**, για την αποθήκευση του στοιχείου, και
- ένα **πεδίο Link**, για την αποθήκευση του δείκτη που δείχνει τη θέση του επόμενου κόμβου μέσα στον πίνακα.

Δηλώσεις -1-

```
#define NumberOfNodes 50 /*μέγεθος της δεξαμενής κόμβων (λίστας*/
#define NilValue 0 /*ειδική μηδενική τιμή, δείχνει το τέλος της Σ.Λ*/
typedef int ListElementType;
/*ο τύπος των στοιχείων της λίστας*/
typedef int ListPointer;        /*ο τύπος των δεικτών*/
typedef struct {
        ListElementType Data;
        ListPointer  Next;
} NodeType;
```

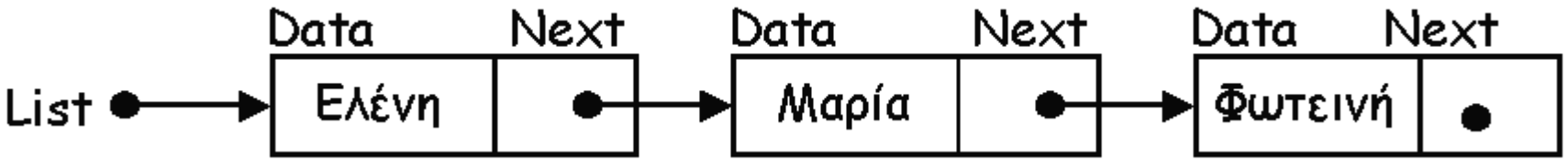
Δηλώσεις -2-

```
/*δήλωση μεταβλητών */

  NodeType Node[NumberOfNodes+1];
        /*η δεξαμενή των διαθέσιμων κόμβων*/
ListPointer FreePtr;
        /*δείκτης για τον πρώτο διαθέσιμο κόμβο*/
```

Παράδειγμα -1-

Ας πάρουμε πάλι τη λίστα ονομάτων:

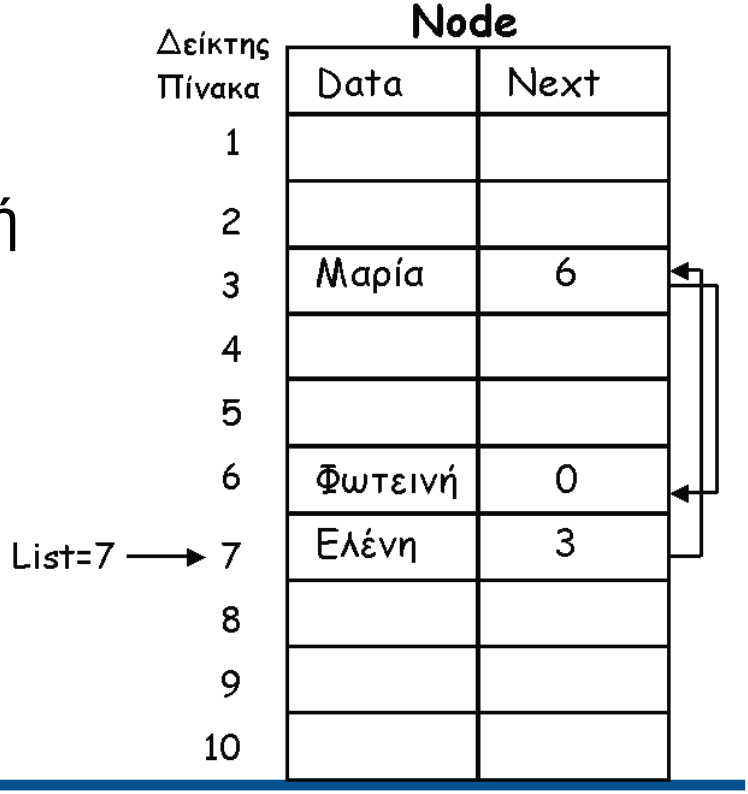


- **List** είναι μια μεταβλητή τύπου LinkedListType και δείχνει στον πρώτο κόμβο, γιατί σ' αυτήν αποθηκεύεται η θέση του στον πίνακα Node.
- Αν υποθέσουμε ότι NumberOfNodes=10, τότε ο πίνακας Node αποτελείται από 10 εγγραφές τύπου NodeType. Οι κόμβοι της ΣΛ μπορούν να είναι αποθηκευμένοι σε οποιεσδήποτε θέσεις του πίνακα αυτού αρκεί οι δεσμοί τους να έχουν τις σωστές τιμές και η List να δείχνει πάντα στον πρώτο κόμβο.

Παράδειγμα -2-

Για παράδειγμα, ο πρώτος κόμβος μπορεί να βρίσκεται στη θέση 7, ο δεύτερος στη θέση 3 και ο τρίτος στη θέση 6, όπως φαίνεται στο παρακάτω σχήμα.

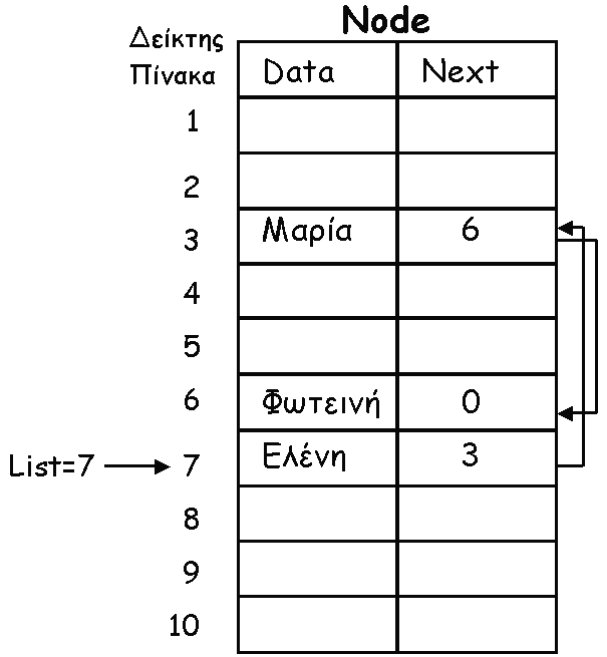
- **List=7.** Στη θέση **Node[7].Data** βρίσκεται το αλφαριθμητικό Ελένη και στη θέση Node[7].Next βρίσκεται η τιμή 3.
- Δεύτερος κόμβος στη θέση **Node[3].Data** βρίσκεται το αλφαριθμητικό Μαρία και στη θέση Node[3].Next η τιμή 6.
- Τρίτο κόμβος **Node[6].Data=Φωτεινή** και Node[6].Next=0, πρόκειται για τον τελευταίο κόμβο που δεν έχει επόμενο και γι' αυτό έχει μηδενικό δείκτη.



Διάσχιση της λίστας

Για να διασχίσουμε τη λίστα και να εμφανίσουμε όλα τα ονόματα με τη σειρά, βρίσκουμε τη θέση του πρώτου κόμβου χρησιμοποιώντας το δείκτη List:

- **List=7**, το πρώτο στοιχείο της λίστας είναι το **Node[7].Data** και εμφανίζεται το όνομα Ελένη.
- Ακολουθώντας το δεσμό του κόμβου αυτού βρίσκουμε ότι το επόμενο στοιχείο βρίσκεται στη θέση **Node[7].Next=3**, δηλαδή είναι το στοιχείο Node[3].Data=Μαρία.
- Το επόμενο στοιχείο της λίστας βρίσκεται στη θέση **Node[3].Next=6** και είναι το όνομα Node[6].Data=Φωτεινή. Η τιμή 0 για το Node[6].Next δείχνει ότι αυτό το στοιχείο είναι το τελευταίο της λίστας.



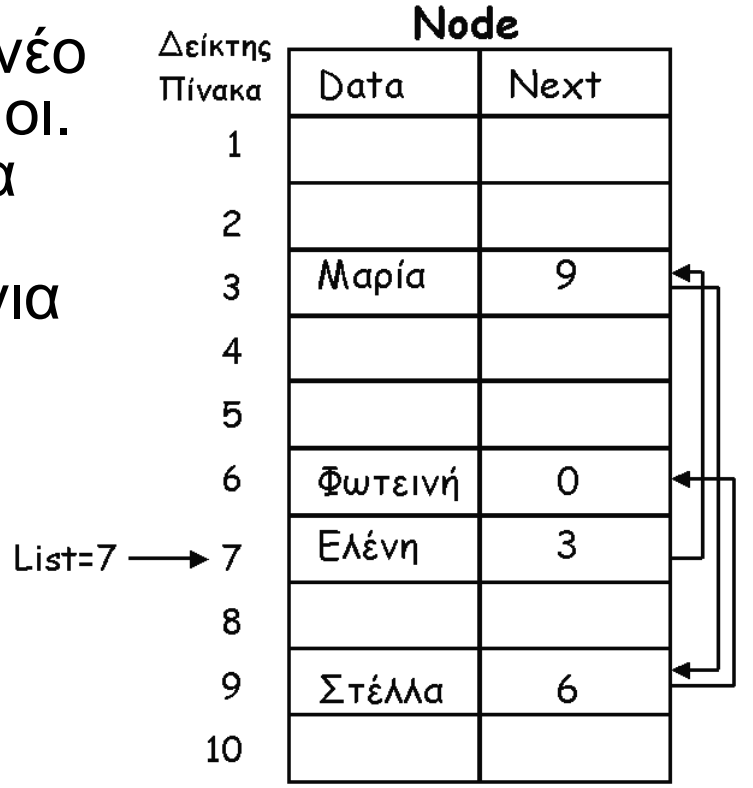
Παράδειγμα (… συνέχεια)

Έστω ότι θέλουμε να εισαγάγουμε το όνομα *Στέλλα* μετά από το όνομα *Μαρία*.

Πρώτα πρέπει να αποκτήσουμε ένα νέο κόμβο από τους 7 που είναι διαθέσιμοι. Υποθέτουμε ότι έχουμε διαθέσιμη μια συνάρτηση **GetNode** η οποία μας επιστρέφει την τιμή 9 ως κενή θέση για το νέο στοιχείο.
Σύμφωνα, λοιπόν, με τη διαδικασία εισαγωγής έχουμε:

Node[9].Data='Στέλλα'
Node[9].Next=6
Node[3].Next=9

και ο πίνακας Node είναι τώρα ο διπλανός:



Δεξαμενή διαθέσιμων κόμβων -1-

Τα στοιχεία του πίνακα Node είναι δύο ειδών:

- σε κάποιες θέσεις υπάρχουν **αποθηκευμένα στοιχεία** της λίστας,
- ενώ οι υπόλοιπες είναι κενές και αποτελούν τις **ελεύθερες θέσεις** για εισαγωγή νέων στοιχείων.

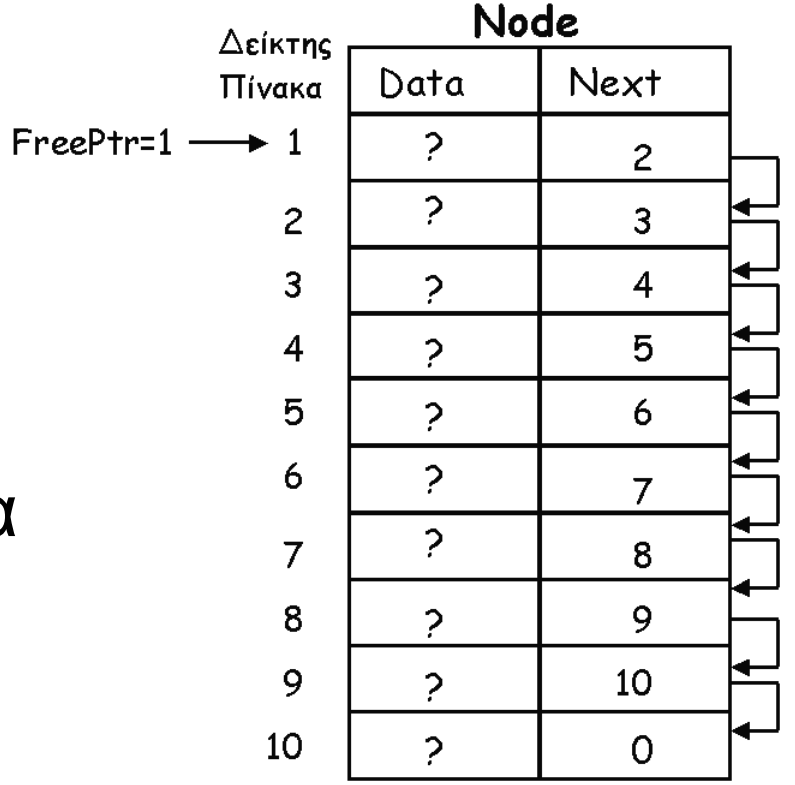
Η οργάνωση των κόμβων που περιέχουν στοιχεία έχει περιγραφεί παραπάνω, μένει, λοιπόν, να περιγράψουμε τον τρόπο οργάνωσης της δεξαμενής των διαθέσιμων κόμβων.

Δεξαμενή διαθέσιμων κόμβων -2-

- Η δεξαμενή μπορεί να οργανωθεί σαν μια ΣΛ.
- Αρχικά όλοι οι κόμβοι είναι διαθέσιμοι, οπότε πρέπει να συνδεθούν μεταξύ τους για να σχηματίσουν τη δεξαμενή. Για να γίνει αυτό μπορούμε να θέσουμε ως πρώτο κόμβο αυτόν που βρίσκεται στην πρώτη θέση, ως δεύτερο κόμβο αυτόν που βρίσκεται στη δεύτερη θέση, κ.ο.κ., κι επομένως, ο πρώτος κόμβος δείχνει στο δεύτερο, ο δεύτερος στον τρίτο, κ.ο.κ. και ο τελευταίος θα έχει μηδενικό δείκτη.
- Τέλος, ένας **δείκτης FreePtr** θα έχει τιμή 1 για να δείχνει στον πρώτο κόμβο.

Αρχικοποίηση της δεξαμενής

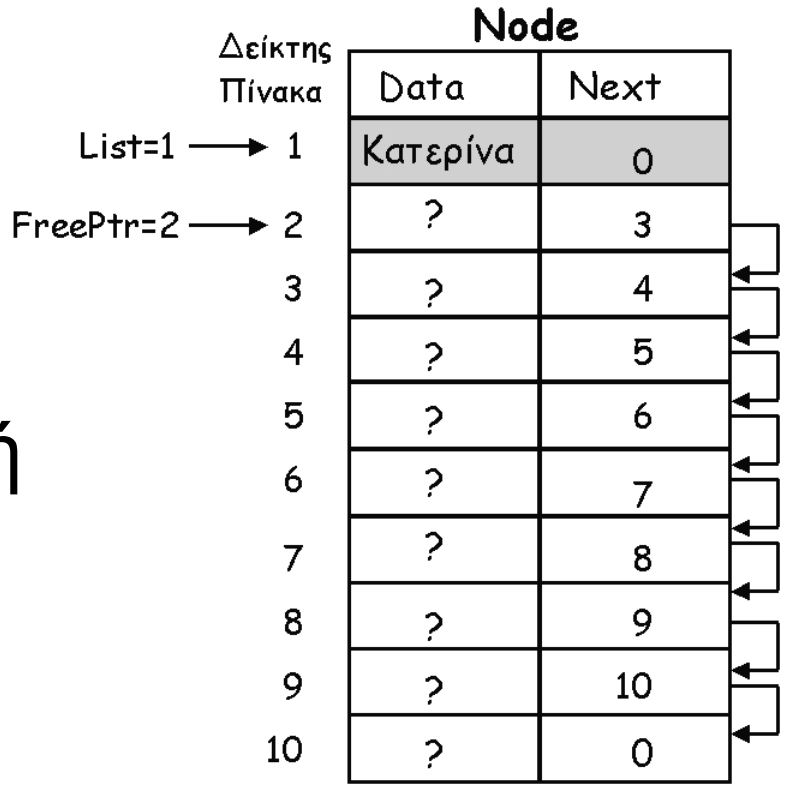
- Ο πίνακας Node θα είναι αρχικά:
- Η κλήση **GetNode(TempPtr)** επιστρέφει τη θέση ενός διαθέσιμου κόμβου θέτοντας TempPtr=FreePtr και διαγράφει αυτόν από τη λίστα των διαθέσιμων κόμβων θέτοντας FreePtr=Node[FreePtr].Next.



Εισαγωγή στοιχείου -1-

Αν το πρώτο στοιχείο που πρόκειται να εισαχθεί είναι το όνομα *Κατερίνα* θα αποθηκευτεί στην πρώτη θέση του πίνακα Node, αφού FreePtr=1.

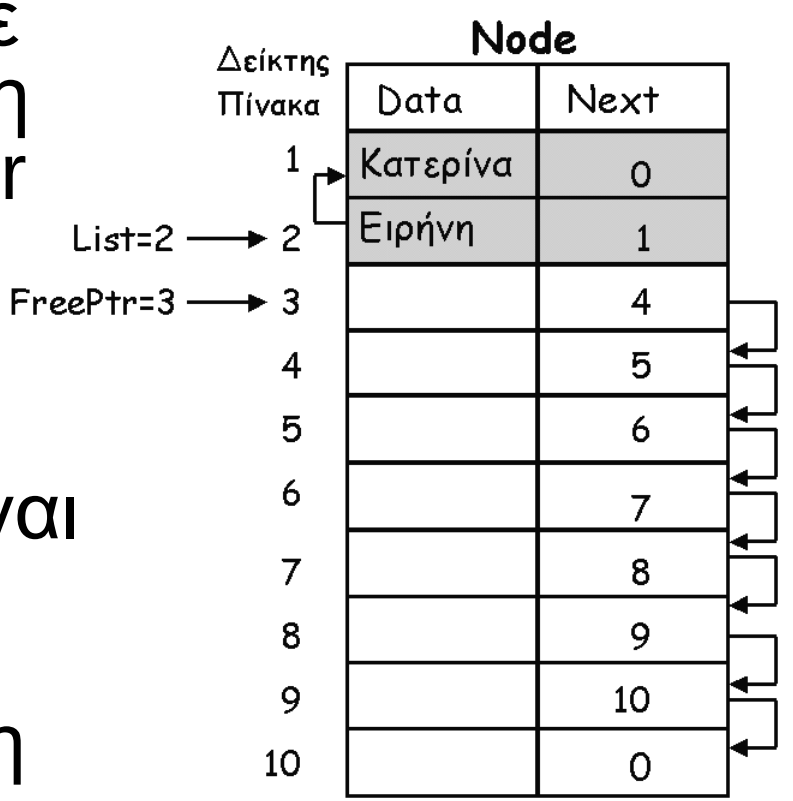
Η μεταβλητή List θα έχει τιμή 1 και η FreePtr θα πάρει τώρα την τιμή 2, όπως φαίνεται παρακάτω:



Εισαγωγή στοιχείου -2-

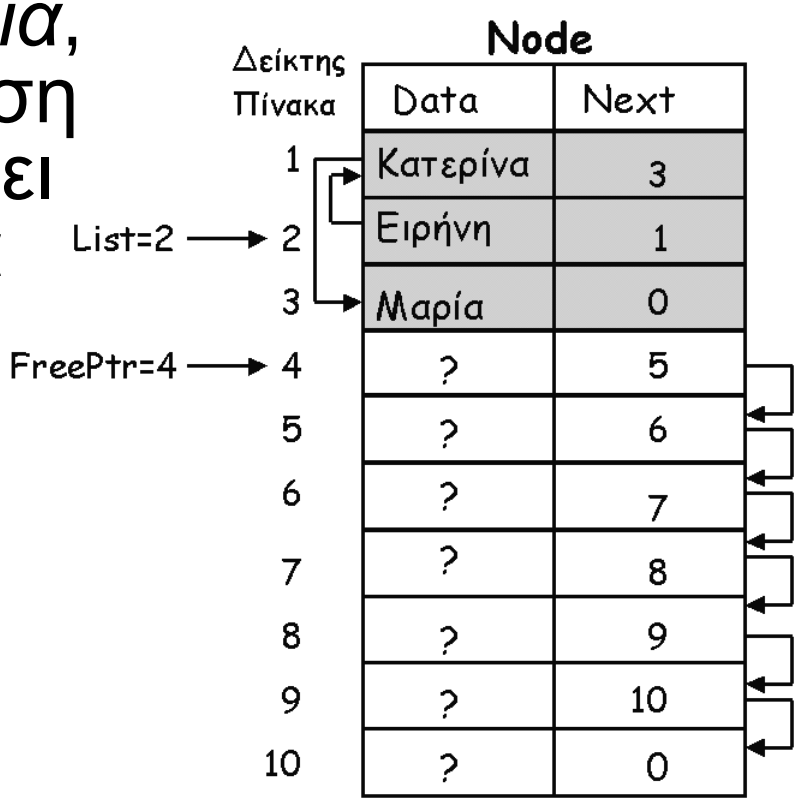
Αν το επόμενο όνομα που θα εισαχθεί είναι το *Ειρήνη*, τότε θα τοποθετηθεί στην δεύτερη θέση, γιατί η τιμή της FreePtr είναι τώρα 2.

Στη συνέχεια η FreePtr θα γίνει ίση με 3 και, αν μας ενδιαφέρει τα ονόματα να είναι με αλφαβητική σειρά, τότε η List θα πάρει την τιμή 2, η Node[2].Next την τιμή 1 και η Node[1].Next την τιμή 0:



Εισαγωγή στοιχείου -3-

Αν τώρα θέλουμε να εισαγάγουμε το όνομα *Μαρία*, τότε θα τοποθετηθεί στη θέση FreePtr=3, η FreePtr θα γίνει ίση με 4, η Node[1].Next θα πάρει την τιμή 3 και η Node[3].Next θα είναι μηδενική:



Διαγραφή κόμβου

Όταν διαγράφουμε έναν κόμβο, τότε αυτός πρέπει να επιστρέψει στη δεξαμενή των διαθέσιμων κόμβων με μια διαδικασία ReleaseNode.

Η κλήση **ReleaseNode(TempPtr)** εισάγει τον κόμβο στον οποίο δείχνει η TempPtr στην αρχή της λίστας των διαθέσιμων κόμβων θέτοντας FreePtr=TempPtr.

Διαγραφή κόμβου -1-

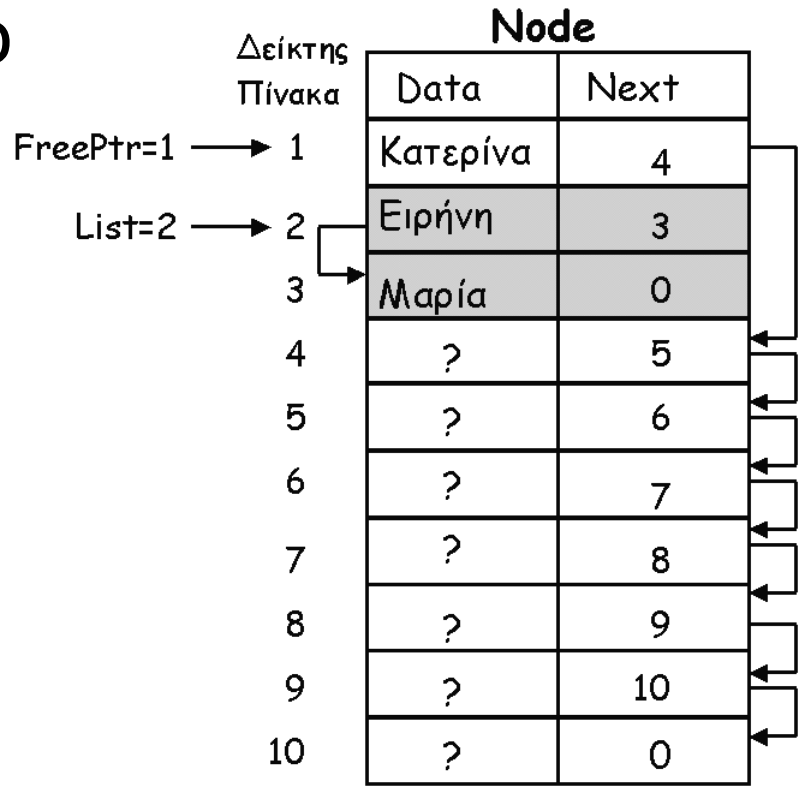
Όταν διαγράφουμε έναν κόμβο, τότε αυτός πρέπει να επιστρέψει στη δεξαμενή των διαθέσιμων κόμβων με μια διαδικασία ReleaseNode.

Η κλήση **ReleaseNode(TempPtr)** εισάγει τον κόμβο στον οποίο δείχνει η TempPtr στην αρχή της λίστας των διαθέσιμων κόμβων θέτοντας FreePtr=TempPtr.

Διαγραφή κόμβου -2-

Αν, για παράδειγμα, θέλουμε να διαγράψουμε το στοιχείο *Κατερίνα*, τότε θέτουμε Node[1].Next=FreePtr και FreePtr=1.

Ο πίνακας Node είναι τώρα όπως φαίνεται δεξιά:



Διαγραφή κόμβου -3-

- Εδώ πρέπει να σημειωθεί ότι **δεν είναι απαραίτητο να διαγράψουμε πραγματικά τη λέξη *Κατερίνα* από τον κόμβο**, γιατί αλλάζοντας το δεσμό του προηγούμενου κόμβου, **αφαιρεί *λογικά* τον κόμβο από την συνδεδεμένη λίστα**.
- Το αλφαριθμητικό *Κατερίνα* θα διαγραφεί πραγματικά όταν θα αποθηκευτεί στη θέση αυτή ένα άλλο αλφαριθμητικό.

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -1-

```
// Filename L_ListADT.h
   #define NumberOfNodes 50
   /*μέγεθος της δεξαμενής κόμβων (λίστας*/

     #define NilValue 0
   /*ειδική μηδενική τιμή, δείχνει το τέλος της Σ.Λ*/

     typedef int ListElementType;
   /*ο τύπος των στοιχείων της ΣΛ*/

     typedef int ListPointer;
   /*ο τύπος των δεικτών*/
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -2-

```
typedef struct {
                ListElementType Data;
        ListPointer  Next;
} NodeType;

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

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -3-

```
void InitializeStoragePool(NodeType Node[ ],
   ListPointer *FreePtr);

void CreateLList(ListPointer *List);

boolean EmptyLList(ListPointer List);

boolean FullLList(ListPointer FreePtr);

void GetNode(ListPointer *P, ListPointer
        *FreePtr, NodeType Node[ ]);

void ReleaseNode(NodeType Node[ ],
        ListPointer P, ListPointer *FreePtr);
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -4-

```
void Insert(ListPointer *List, NodeType Node[ ],
   ListPointer *FreePtr, ListPointer PredPtr,
   ListElementType Item);

void Delete(ListPointer *List, NodeType Node[],
   ListPointer *FreePtr, ListPointer PredPtr);

void TraverseLinked(ListPointer List,
        NodeType Node[ ]);
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -5-

void InitializeStoragePool(NodeType Node[ ], ListPointer *FreePtr);

/*Δέχεται: Τον πίνακα *Node* και τον δείκτη *FreePtr* που δείχνει στον πρώτο διαθέσιμο κόμβο.

Λειτουργία: Αρχικοποιεί τον πίνακα *Node* ως συνδεδεμένη λίστα συνδέοντας μεταξύ τους διαδοχικές εγγραφές του πίνακα, και αρχικοποιεί τον δείκτη *FreePtr* .

Επιστρέφει: Τον τροποποιημένο πίνακα *Node* και τον δείκτη *FreePtr* του πρώτου διαθέσιμου κόμβου.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -6-

```
{
   int i;
    for (i=1; I < NumberOfNodes; i++) {
        Node[i].Next = i+1;
        Node[i].Data = -1;        /* δεν είναι αναγκαίο η
         απόδοση αρχικής τιμής στο πεδίο των δεδομένων */
    }
    Node[NumberOfNodes].Next = NilValue ;
    Node[NumberOfNodes].Data = -1;
    *FreePtr = 1;
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -7-

```
{
   int i;
    for (i=1; I < NumberOfNodes; i++) {
        Node[i].Next = i+1;
        Node[i].Data = -1;        /* δεν είναι αναγκαίο η
         απόδοση αρχικής τιμής στο πεδίο των δεδομένων */
    }
    Node[NumberOfNodes].Next = NilValue ;
    Node[NumberOfNodes].Data = -1;
    *FreePtr = 1;
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -8-

```
void CreateLList(ListPointer *List)
/*Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη
   λίστα.
   Επιστρέφει: Έναν (μηδενικό) δείκτη που
   δείχνει σε κενή ΣΛ.*/
{
   *List = NilValue;
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -9-

```
boolean EmptyLList(ListPointer List);
/*Δέχεται:  Έναν δείκτη List που δείχνει στο 1ο
   στοιχείο της συνδεδεμένης λίστας.
   Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα
   είναι κενή.
   Επιστρέφει: TRUE αν η συνδεδεμένη λίστα
   είναι κενή και FALSE διαφορετικά.*/
{
   return (List == NilValue);
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -10-

```
boolean FullLList(ListPointer FreePtr)
/* Δέχεται:  Έναν δείκτη FreePtr που δείχνει σε μια
   συνδεδεμένη λίστα με διαθέσιμους κόμβους.
   Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι
   γεμάτη.
   Επιστρέφει: TRUE αν η συνδεδεμένη λίστα είναι
   γεμάτη και FALSE διαφορετικά.*/
{
   return (FreePtr == NilValue);
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -11-

void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[ ])

/*Δέχεται: Τον πίνακα *Node* με τα στοιχεία της ΣΛ και τους διαθέσιμους κόμβους και τον δείκτη *FreePtr*.

Λειτουργία: Αποκτά τον 1ο "ελεύθερο" κόμβο.

Επιστρέφει: Τον δείκτη *P* πο δείχνει στο διαθέσιμο κόμβο και τον τροποποιημένο δείκτη FreePtr που δεικτοδοτεί στο 1ο (νέο) διαθέσιμο κόμβο.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -12-

```
{
   *P = *FreePtr;
   if (!FullLList(*FreePtr))
        *FreePtr = Node[*FreePtr].Next;
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -13-

void Insert(ListPointer *List, NodeType Node[ ], ListPointer *FreePtr, ListPointer PredPtr, ListElementType Item);

/*Δέχεται: Μια συνδεδεμένη λίστα (δείκτη *List* προς το 1ο στοιχείο της ΣΛ, τον πίνακα *Node* με τα στοιχεία της ΣΛ), τον δείκτη *PredPtr* και ένα στοιχείο *Item*.

Λειτουργία: Εισάγει στη συνδεδεμένη λίστα, αν δεν είναι γεμάτη, το στοιχείο *Item* μετά από τον κόμβο στον οποίο δείχνει ο δείκτης *PredPtr*.

Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα: δείκτη *List* προς το 1ο στοιχείο της ΣΛ, τον τροποποιημένο πίνακα *Node* και τον δείκτη *FreePtr* προς το 1ο διαθέσιμο κόμβο

Εξοδος: Μήνυμα γεμάτης λίστας, αν η συνδεδεμένη λίστα είναι γεμάτη.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -14-

```
{
    ListPointer TempPtr;
    // η GetNode επιστρέφει τη θέση TempPtr που θα αποθηκευθεί το στοιχείο
    GetNode(&TempPtr, FreePtr, Node);
    if (!FullLList(TempPtr)) {
        if (PredPtr == NilValue) {
            Node[TempPtr].Data =Item;
            Node[TempPtr].Next =*List;
            *List =TempPtr; }
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -15-

```
else  {
        Node[TempPtr].Data = Item;
        Node[TempPtr].Next = Node[PredPtr].Next;
        Node[PredPtr].Next = TempPtr;
         }
     }
     else
          printf("Full List ...\n");
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -16-

void Delete(ListPointer *List, NodeType Node[ ], ListPointer *FreePtr, ListPointer PredPtr);

/*Δέχεται: Μια συνδεδεμένη λίστα (δείκτη *List* προς το 1ο στοιχείο της ΣΛ, τον πίνακα *Node* με τα στοιχεία της ΣΛ), και τον δείκτη *PredPtr* που δείχνει στον προηγούμενο κόμβο από αυτόν που θα διαγραφεί.

Λειτουργία: Διαγράφει από τη συνδεδεμένη λίστα, αν δεν είναι κενή, τον επόμενο κόμβο από αυτόν στον οποίο δείχνει ο *PredPtr*.

Επιστρέφει: Την τροποποιημένη λίστα και το δείκτη *FreePtr* προς το 1ο διαθέσιμο κόμβο (θέση του στοιχείου που διαγράφτηκε)

Έξοδος: Μήνυμα κενής λίστας, αν η συνδεδεμένη λίστα είναι κενή.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -17-

```
{
        ListPointer TempPtr ;
        if (!EmptyLList(*List)) {
            if (PredPtr == NilValue)  {
                TempPtr =*List;
                *List = Node[TempPtr].Next;
             }
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -18-

```
else {
        TempPtr = Node[PredPtr].Next;
        Node[PredPtr].Next = Node[TempPtr].Next;
    }
    ReleaseNode(Node, TempPtr, &(*FreePtr));
}
else
        printf("Empty List ...\n");
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -19-

void ReleaseNode(NodeType Node[ ], ListPointer P, ListPointer *FreePtr);

/*Δέχεται: Τον πίνακα *Node*, που αναπαριστά τα στοιχεία της ΣΛ και τη δεξαμενή των διαθέσιμων κόμβων, και έναν δείκτη *P*.

Λειτουργία: Επιστρέφει στη δεξαμενή τον κόμβο στον οποίο δείχνει ο *P*.

Επιστρέφει: Τον τροποποιημένο πίνακα *Node* και τον δείκτη *P*.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -20-

```
{
   Node[P].Next =*FreePtr;
   *FreePtr = P;
   Node[P].Data = -1;
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -21-

void TraverseLinked(ListPointer List, NodeType Node[ ]);

/*Δέχεται: Μια συνδεδεμένη λίστα (δείκτη *List* προς το 1ο στοιχείο της ΣΛ, τον πίνακα *Node* με τα στοιχεία της ΣΛ).

Λειτουργία: Κάνει διάσχιση της συνδεδεμένης λίστας, αν δεν είναι κενή.

Έξοδος: Εξαρτάται από την επεξεργασία.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -21-

```
{
     ListPointer CurrPtr;
    if (!EmptyLList(List)) {
       CurrPtr =List;
       while (CurrPtr != NilValue) {
              printf("(%d, %d, %d) ",
              CurrPtr,Node[CurrPtr].Data,
              Node[CurrPtr].Next);
              CurrPtr = Node[CurrPtr].Next;
       }
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα -22-

```
    printf("\n");
   }
  else printf("Empty List ...\n");
}
```

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