

ΛΙΣΤΕΣ

4.3 Υλοποίηση ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

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**.

Οι κόμβοι μιας συνδεδεμένης λίστας περιλαμβάνουν:

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

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 Next**, για την αποθήκευση του αριθμοδείκτη που δείχνει τη θέση του επόμενου κόμβου μέσα στον πίνακα.

```
#define NumberOfNodes 50      /*μέγεθος της δεξαμενής κόμβων (λίστας)*/

#define NilValue -1           /*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;

/*δήλωση μεταβλητών */

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

Παράδειγμα

Ας πάρουμε πάλι τη λίστα ονομάτων:



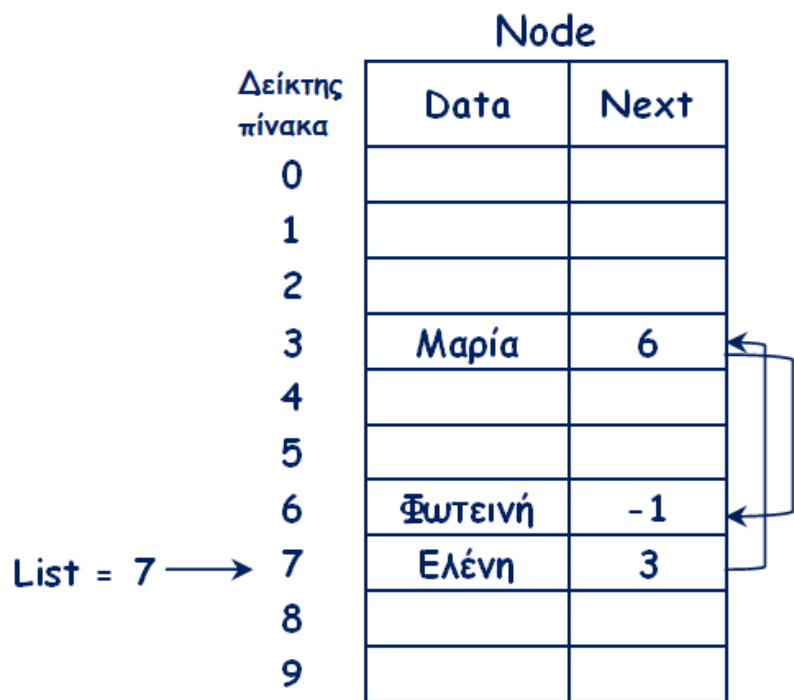
List είναι μια μεταβλητή τύπου `ListPointer` και δείχνει στον πρώτο κόμβο, γιατί σ' αυτήν αποθηκεύεται η θέση του στον πίνακα `Node`.

Αν υποθέσουμε ότι `NumberOfNodes=10`, τότε ο πίνακας `Node` αποτελείται από 10 εγγραφές τύπου `NodeType`.

Οι κόμβοι της συνδεδεμένης λίστας 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

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.
- Ομοίως για το δεύτερο κόμβο, στη θέση **Node[3].Data** βρίσκεται το αλφαριθμητικό Μαρία και στη θέση **Node[3].Next** η τιμή 6.
- Τέλος, για τον τρίτο κόμβο έχουμε **Node[6].Data=Φωτεινή** και **Node[6].Next=-1**, αφού πρόκειται για τον τελευταίο κόμβο ο οποίος δεν έχει επόμενο και γι' αυτό έχει μηδενικό αριθμοδείκτη.


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 αριθμοδείκτη List:

- Αφού **List=7**, το πρώτο στοιχείο της λίστας είναι το **Node[7].Data** και επομένως 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**.
- Ομοίως, το επόμενο στοιχείο της λίστας βρίσκεται στη θέση **Node[3].Next=6** και είναι το όνομα **Node[6].Data=Φωτεινή**. Η τιμή -1 για το **Node[6].Next** δείχνει ότι αυτό το στοιχείο είναι το τελευταίο της λίστας.

Δείκτης πίνακα	Node	
	Data	Next
0		
1		
2		
3	Μαρία	6
4		
5		
6	Φωτεινή	-1
7	Ελένη	3
8		
9		

List = 7 →



Διάσχιση Συνδεδεμένης Λίστας TraverseLinked()

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

*/*Δέχεται: Μια συνδεδεμένη λίστα (αριθμοδείκτη *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./**

{

ListPointer CurrPtr;

if (!EmptyList(List))

{

CurrPtr = List;

while (CurrPtr != NilValue) {

printf("(%d, %d, %d) ", CurrPtr, Node[CurrPtr].Data,

Node[CurrPtr].Next);

CurrPtr = Node[CurrPtr].Next;

}

printf("\n");

}

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

}

*/*Σειριακή διάσχιση της ΣΛ. Κάθε φορά προσπελαύνουμε (γίνεται τρέχον *CurrPtr*) το επόμενο στοιχείο του τρέχοντος (λογική όχι φυσική διάταξη) Node[CurrPtr].Next*/*

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:

Node	
Δείκτης πίνακα	
	Data
	Next
0	
1	
2	
3	Μαρία
4	
5	
6	Φωτεινή
7	Ελένη
8	
9	Στέλλα

List = 7 →

```
graph TD; 3 -- 9 --> 9; 6 -- -1 --> 6; 9 -- 6 --> 6;
```

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.

Η δεξαμενή διαθέσιμων κόμβων

- Η δεξαμενή μπορεί να οργανωθεί σαν μια συνδεδεμένη λίστα.
- Αρχικά όλοι οι κόμβοι είναι διαθέσιμοι, οπότε πρέπει να συνδεθούν μεταξύ τους για να σχηματίσουν τη δεξαμενή.
- Για να γίνει αυτό μπορούμε πολύ απλά 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 αριθμοδείκτη.
- Τέλος, ένας **αριθμοδείκτης FreePtr** θα έχει τιμή 0 για να δείχνει στον πρώτο κόμβο.

Αρχικοποίηση της δεξαμενής InitializeStoragePool()

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

*/** Δέχεται: Τον πίνακα *Node* και τον αριθμοδείκτη *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 αριθμοδείκτη *FreePtr* .

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

int i;

for (i=0; i < NumberOfNodes-1; i++)

{

Node[i].Next = i+1;

Node[i].Data = -1; */* δεν είναι αναγκαίο η απόδοση αρχικής τιμής στο πεδίο των δεδομένων */*

}

Node[NumberOfNodes-1].Next = NilValue ;

Node[NumberOfNodes-1].Data = -1;

*FreePtr = 0;

}

Διαδικασίες δημιουργίας & ελέγχου κενής λίστας

Οι διαδικασίες δημιουργίας κενής λίστας και ελέγχου αν μια συνδεδεμένη λίστα είναι κενή είναι απλές:

void CreateList(ListPointer *List)

/*Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη λίστα.

Επιστρέφει: Έναν (μηδενικό) αριθμοδείκτη που δείχνει σε κενή ΣΛ.*/

```
{  
    *List = NilValue;  
}
```

boolean EmptyList(ListPointer List)

/*Δέχεται: Έναν αριθμοδείκτη *List* που δείχνει στο 1^ο στοιχείο της ΣΛ.

Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.

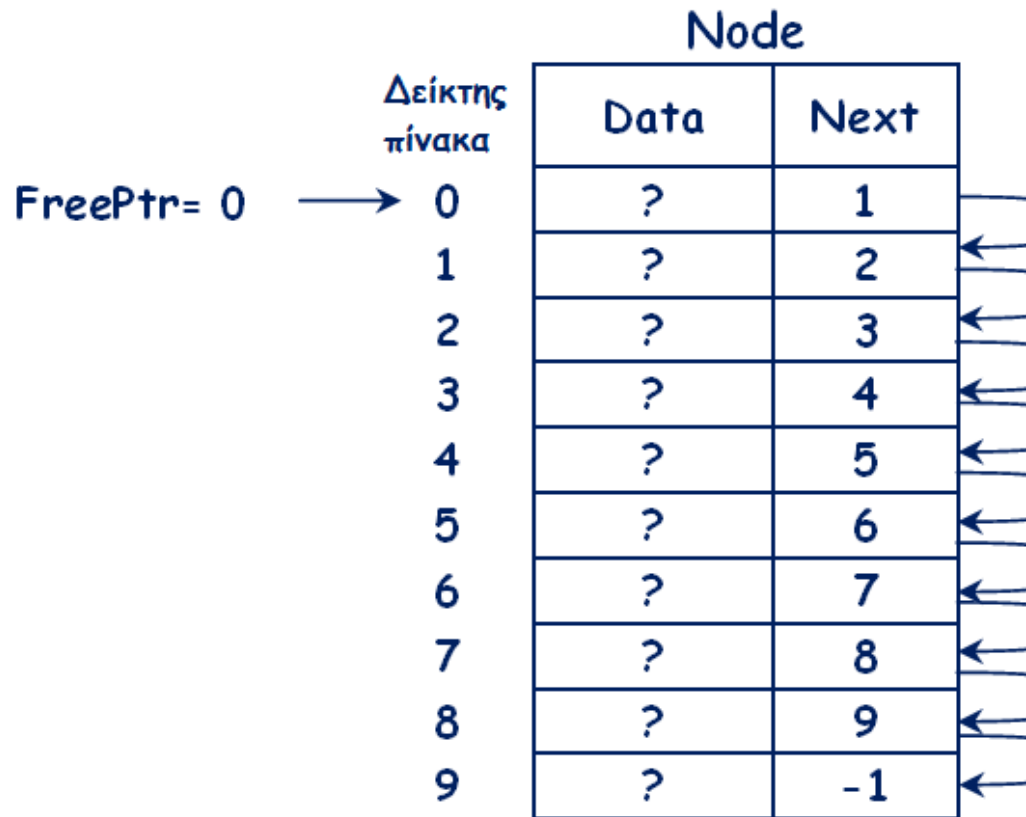
Επιστρέφει: TRUE αν η συνδεδεμένη λίστα είναι κενή και FALSE διαφορετικά.*/

```
{  
    return (List == NilValue);  
}
```

Αρχικοποίηση της δεξαμενής

Ο πίνακας Node θα είναι αρχικά:

		Node	
		Data	Next
FreePtr= 0	→ 0	?	1
	1	?	2
	2	?	3
	3	?	4
	4	?	5
	5	?	6
	6	?	7
	7	?	8
	8	?	9
	9	?	-1



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.

Ανάκτηση ελεύθερου κόμβου GetNode()

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

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

Λειτουργία: Αποκτά τον 1ο "ελεύθερο" κόμβο.

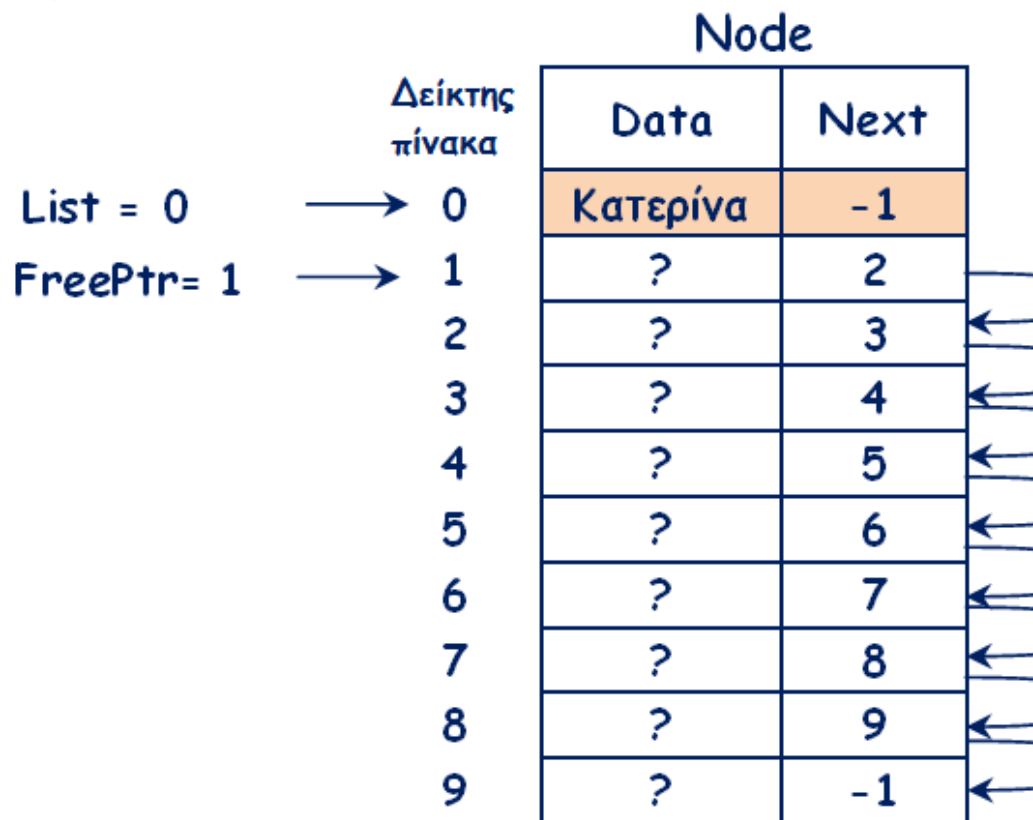
Επιστρέφει: Τον αριθμοδείκτη *P* που δείχνει στο διαθέσιμο κόμβο και τον τροποποιημένο αριθμοδείκτη *FreePtr* που δεικτοδοτεί στο 1^ο (νέο) διαθέσιμο κόμβο.**/*

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

Εισαγωγή στοιχείου

Αν το πρώτο στοιχείο που πρόκειται να εισαχθεί είναι το όνομα *Κατερίνα* θα αποθηκευτεί στην πρώτη θέση του πίνακα Node, αφού FreePtr=0.

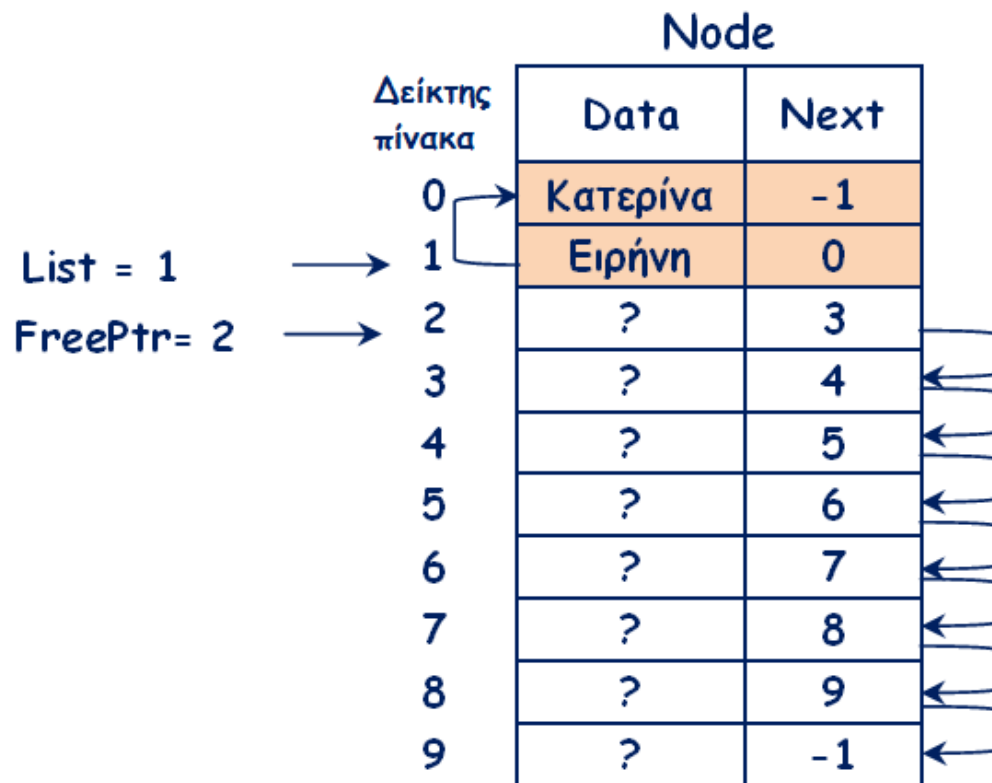
Η μεταβλητή List θα έχει τιμή 0 και η FreePtr θα πάρει τώρα την τιμή 1, όπως φαίνεται παρακάτω:



Εισαγωγή στοιχείου

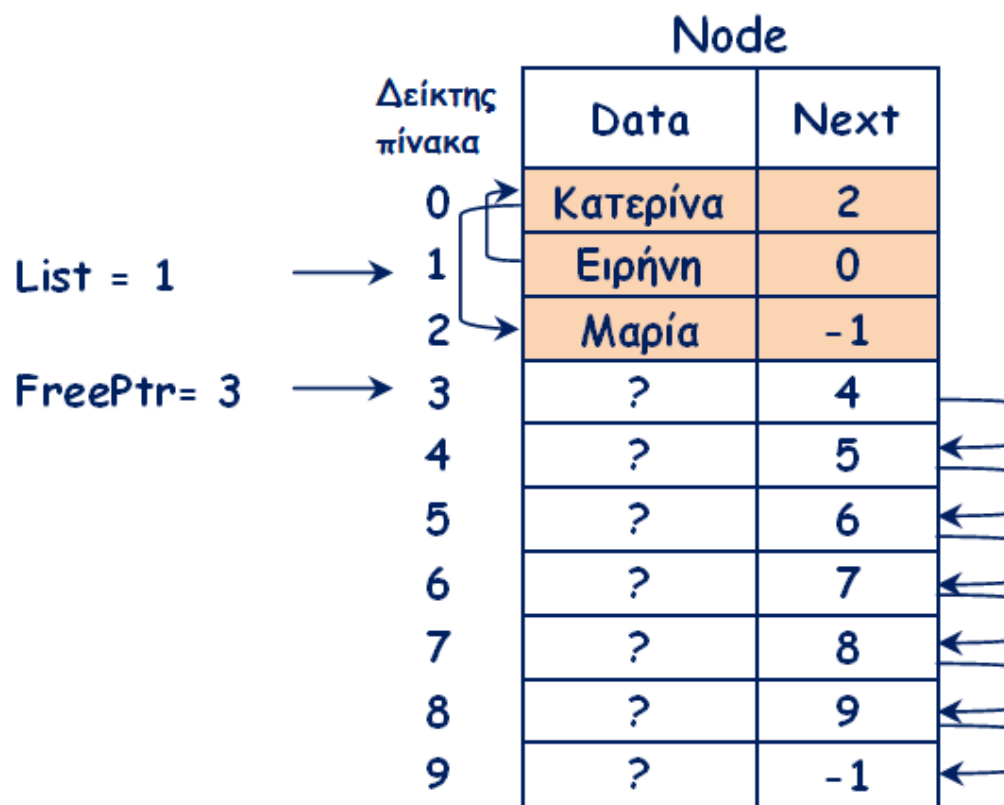
Αν το επόμενο όνομα που θα εισαχθεί είναι το *Ειρήνη*, τότε θα τοποθετηθεί στην δεύτερη θέση, γιατί η τιμή της FreePtr είναι τώρα 1.

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



Εισαγωγή στοιχείου

Αν τώρα θέλουμε να εισαγάγουμε το όνομα *Μαρία*, τότε θα τοποθετηθεί στη θέση $\text{FreePtr}=2$, η FreePtr θα γίνει ίση με 3, η $\text{Node}[0].\text{Next}$ θα πάρει την τιμή 2 και η $\text{Node}[2].\text{Next}$ θα είναι -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 ReleaseNode()

```
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 αριθμοδείκτη P.*

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

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

```
{
```

```
    Node[P].Next = *FreePtr; /*ο κόμβος που διαγράφηκε προστέθηκε στην αρχή της ΣΛ των διαθέσιμων κόμβων */
```

```
    *FreePtr = P;
```

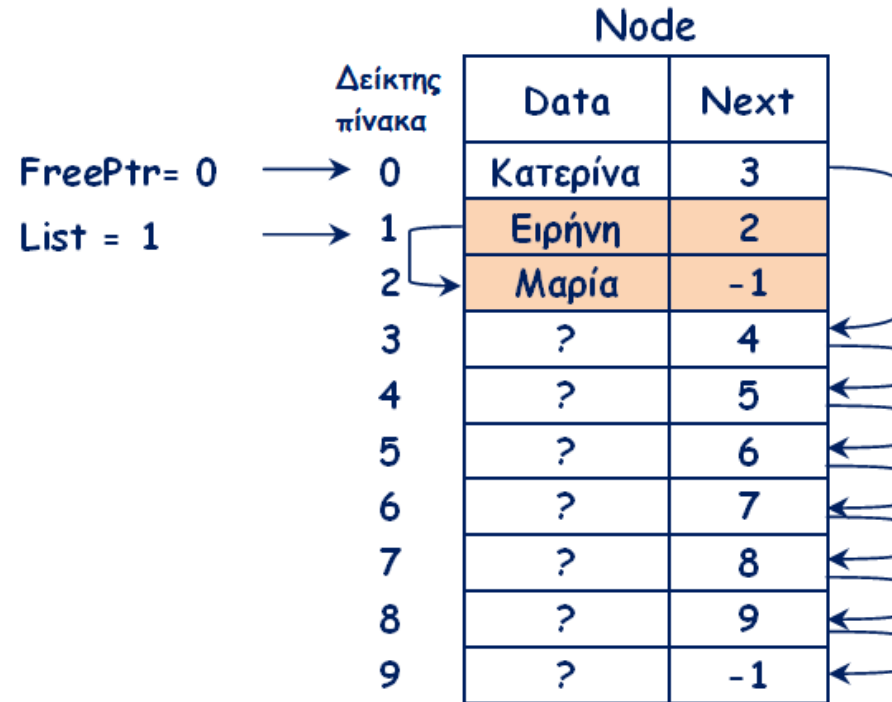
```
    Node[P].Data = -1; /*Οχι αναγκαία εντολή, βοηθητική για να φαίνονται στην εμφάνιση οι διαγραφμένοι κόμβοι */
```

```
}
```

Διαγραφή κόμβου

Αν, για παράδειγμα, θέλουμε να διαγράψουμε το στοιχείο *Κατερίνα*, τότε θέτουμε $\text{Node}[0].\text{Next} = \text{FreePtr}$ και $\text{FreePtr} = 0$.

Ο πίνακας *Node* είναι τώρα όπως φαίνεται δεξιά:



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

// Filename L_ListADT.h

```
#define NumberOfNodes 50      /*μέγεθος της δεξαμενής κόμβων (λίστας)*/  
#define NilValue -1          /*ειδική μηδενική τιμή, δείχνει το τέλος της Σ.Λ*/
```

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

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

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

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

```
main() {  
    // δηλώσεις μεταβλητών  
    ListPointer AList; // αριθμοδείκτης προς το 1ο στοιχείο της ΣΛ  
    NodeType Node[NumberOfNodes]; //πίνακας με τα  
    στοιχεία της ΣΛ & της δεξαμενής των ελεύθερων κόμβων  
    ListPointer FreePtr, PredPtr; // δείκτες προς το 1ο  
    διαθέσιμο κόμβο στη δεξαμενή ελεύθερων κόμβων,  
    προς το προηγούμενο για εισαγωγή  
    ListElementType AnItem; // το στοιχείο που θα εισαχθεί  
  
    .....
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

```
void InitializeStoragePool(NodeType Node[ ], ListPointer *FreePtr);  
void CreateList(ListPointer *List);  
boolean EmptyList(ListPointer List);  
boolean FullList(ListPointer FreePtr);  
void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[ ]);  
void ReleaseNode(NodeType Node[ ], ListPointer P, ListPointer *FreePtr);  
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

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

*/** Δέχεται: Τον πίνακα *Node* και τον αριθμοδείκτη *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 αριθμοδείκτη *FreePtr* .

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

{

int i;

for (i=0; i < NumberOfNodes-1; i++)

{

Node[i].Next = i+1;

Node[i].Data = -1; */* δεν είναι αναγκαίο η απόδοση αρχικής τιμής στο πεδίο των δεδομένων */*

}

Node[NumberOfNodes-1].Next = NilValue ;

Node[NumberOfNodes-1].Data = -1;

*FreePtr = 0;

}

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

```
void CreateList(ListPointer *List);
```

/*Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη λίστα.

Επιστρέφει: Έναν (μηδενικό) αριθμοδείκτη που δείχνει σε κενή λίστα.*/*

```
{
```

```
    *List = NilValue;
```

```
}
```

```
boolean EmptyList(ListPointer List);
```

/*Δέχεται: Έναν αριθμοδείκτη *List* που δείχνει στο 1^ο 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

```
boolean FullList(ListPointer FreePtr)
```

/ Δέχεται: Έναν αριθμοδείκτη *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

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

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

Λειτουργία: Αποκτά τον 1ο "ελεύθερο" κόμβο.

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

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

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

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

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

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

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



Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

```
{
    ListPointer TempPtr;
    //   η GetNode επιστρέφει τη θέση TempPtr που θα αποθηκευθεί το στοιχείο
    GetNode(&TempPtr, FreePtr, Node);
    if (!FullList(TempPtr)) {
        if (PredPtr == NilValue)
            /* το στοιχείο εισάγεται στην αρχή της ΣΛ
            (δεν έχει προηγούμενο), επόμενο του θα
            γίνει το μέχρι τώρα 1ο στοιχείο της ΣΛ */
            {
                Node[TempPtr].Data = Item;
                Node[TempPtr].Next = *List;
                *List = TempPtr;
            }
        else
            /* το στοιχείο έχει προηγούμενο, εισαγωγή
            μετά απ' αυτό και θα έχει επόμενο το επόμενο
            του προηγούμενου του */
            {
                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

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

*/** Δέχεται: Μια συνδεδεμένη λίστα (αριθμοδείκτη *List* προς το 1ο στοιχείο της πίνακα *Node* με τα στοιχεία της ΣΛ), και τον αριθμοδείκτη *PredPtr* 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 αριθμοδείκτη *FreePtr* προς το 1ο διαθέσιμο κόμβο (θέση του στοιχείου που διαγράφηκε)

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



Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

```
{
    ListPointer TempPtr ;

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

/ το στοιχείο που διαγράφεται είναι το 1ο,
(δεν έχει προηγούμενο) */*

/ το στοιχείο που διαγράφεται έχει
προηγούμενο, το προηγούμενο του θα
έχει επόμενο το επόμενο του στοιχείου
που διαγράφεται */*

/ Ο κόμβος που διαγράφεται
δεικτοδοτείται από το αριθμοδείκτη
TempPtr, αποδίδεται στη δεξαμενή
των διαθέσιμων κόμβων */*

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

```
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 αριθμοδείκτη *P*.*

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

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

```
{
```

```
    Node[P].Next = *FreePtr; /*ο κόμβος που διαγράφηκε προστέθηκε στην αρχή της ΣΛ των διαθέσιμων κόμβων */
```

```
    *FreePtr = P;
```

```
    Node[P].Data = -1; /*Οχι αναγκαία εντολή, βοηθητική για να φαίνονται στην εμφάνιση οι διαγραφμένοι κόμβοι */
```

```
}
```


Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

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

*/*Δέχεται: Μια συνδεδεμένη λίστα (αριθμοδείκτη *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./**

```
{
```

```
    ListPointer CurrPtr;
```

```
    if (!EmptyList(List))
```

```
    {
```

```
        CurrPtr = List;
```

```
        while (CurrPtr != NilValue) {
```

```
            printf("(%d, %d, %d) ", CurrPtr, Node[CurrPtr].Data,
```

```
            Node[CurrPtr].Next);
```

```
            CurrPtr = Node[CurrPtr].Next;
```

```
        }
```

```
        printf("\n");
```

```
    }
```

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

```
}
```

*/*Σειριακή διάσχιση της ΣΛ. Κάθε φορά προσπελαύνουμε (γίνεται τρέχον *CurrPtr*) το επόμενο στοιχείο του τρέχοντος (λογική όχι φυσική διάταξη) *Node[CurrPtr].Next**/*

Παράδειγμα χρήσης του ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

Ως ένα παράδειγμα της χρήσης του ΑΤΔ Συνδεδεμένη Λίστα με πίνακα, κατασκευάστηκε το πρόγραμμα `Reverse1.c` (περιλαμβάνεται στο `Project_Reverse1`) που εμφανίζει το ανάστροφο ενός συνόλου χαρακτήρων.