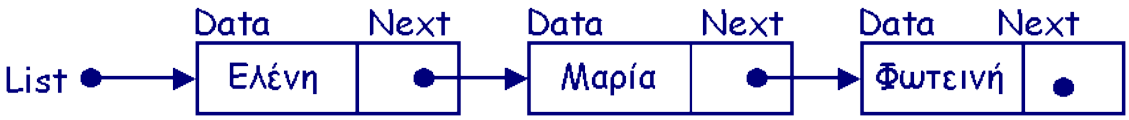
4.3 Υλοποίηση ΑΤΔ Συνδεδεμένη Λίστα με πίνακα

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, όπως κάναμε με τις στοίβες και τις ουρές.

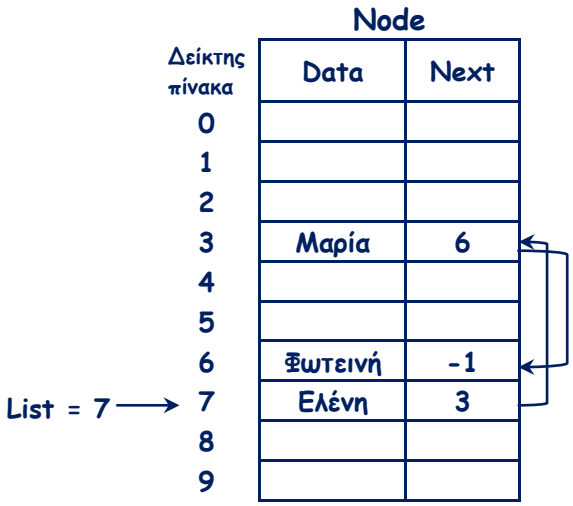
Οι κόμβοι μιας συνδεδεμένης λίστας περιλαμβάνουν το *τμήμα δεδομένου* (Data), όπου αποθηκεύεται ένα στοιχείο της λίστας, και το *τμήμα δεσμού* (Link), όπου αποθηκεύεται ένας δείκτης, ο οποίος είτε δείχνει στον επόμενο κόμβο της λίστας είτε είναι μηδενικός, στην περίπτωση που είναι ο δείκτης του τελευταίου κόμβου. Επομένως, 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. Οι απαραίτητες δηλώσεις για την υλοποίηση των συνδεδεμένων λιστών με πίνακα φαίνονται παρακάτω:

```
#define NumberOfNodes 50                  /* μέγεθος της δεξαμενής */
#define NilValue -1                        /* ειδική μηδενική τιμή */
typedef int ListElementType;               /* ο τύπος των στοιχείων της
                                              λίστας, ενδεικτικά τύπου int */
typedef int ListPointer;                   /* ο τύπος των δεικτών */
typedef struct {
            ListElementType Data;
            ListPointer  Next;
} NodeType;
     NodeType Node[NumberOfNodes];       /* η δεξαμενή των διαθέσιμων κόμβων */
     ListPointer FreePtr;                /*δείκτης για τον πρώτο διαθέσιμο κόμβο*/
     ListPointer AList;                   /* δείκτης για τον πρώτο κόμβο της
                                             συνδεδεμένης λίστας*/
```

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. 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.



Επομένως, 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 δείκτη.

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

Ο αλγόριθμος για την διάσχιση μιας συνδεδεμένης λίστας είναι αυτός που περιγράφεται στην ενότητα 4.2, ενώ σε μορφή κώδικα προγράμματος φαίνεται παρακάτω:

TRAVERSE

```
/*Δέχεται:        Μια συνδεδεμένη λίστα.
  Λειτουργία:    Κάνει διάσχιση της συνδεδεμένης λίστας, αν δεν είναι
                 κενή.
```

```
  Έξοδος:          Εξαρτάται από την επεξεργασία.*/
{
  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;
       }
       printf("\n");
   }
  else printf("Empty List ...\n");
}
```

όπου η μεταβλητή CurrPtr είναι τύπου ListPointer.

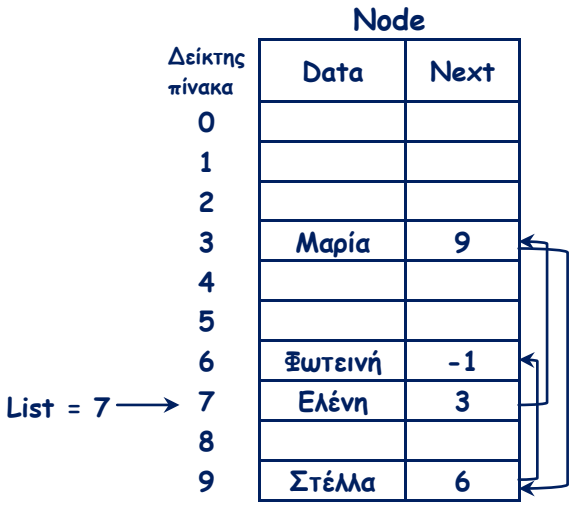
Έστω τώρα ότι θέλουμε να εισαγάγουμε το όνομα *Στέλλα* μετά από το όνομα *Μαρία*. Πρώτα πρέπει να αποκτήσουμε ένα νέο κόμβο από τους 7 που είναι διαθέσιμοι. Υποθέτουμε ότι έχουμε διαθέσιμη μια συνάρτηση GetNode η οποία μας επιστρέφει την τιμή 9 ως κενή θέση για το νέο στοιχείο. Σύμφωνα, λοιπόν, με τη διαδικασία εισαγωγής που περιγράφεται στην ενότητα 4.2, έχουμε:

Node[9].Data='Στέλλα'

Node[9].Next=6

Node[3].Next=9

και ο πίνακας Node είναι τώρα ο παρακάτω:



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

InitializeStoragePool

/*Δέχεται: Έναν πίνακα *Node* από *NumberOfNodes* εγγραφές.

Λειτουργία: Αρχικοποιεί τη δεξαμενή *Node* σαν συνδεδεμένη λίστα συνδέοντας τις εγγραφές με τη σειρά.

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

/*Ενημέρωση του πίνακα *Node* ώστε κάθε στοιχείο του, από το πρώτο μέχρι το προτελευταίο, να δείχνει στο αμέσως επόμενο στοιχείο του πίνακα, δημιουργώντας έτσι μια συνδεδεμένη λίστα*/

1. **Για** *i* **από** 0 **μέχρι** *NumberOfNodes*-2

   Node[*i*].*Next* ← *i* + 1

   /*Ενημέρωσε το πεδίο *Next* του *i* στοιχείου του πίνακα *Node,* έτσι ώστε να δείχνει στο αμέσως επόμενο στοιχείο του πίνακα, δηλαδή στο στοιχείο *i* + 1*/

   Τέλος_επανάληψης

2. *Node*[*NumberOfNodes*-1].*Next* ← -1

   /*Θέσε στο πεδίο *Next* του τελευταίου στοιχείου του πίνακα *Node*, δηλαδή του στοιχείου που βρίσκεται στη θέση *NumberOfNodes* -1, την τιμή -1 αφού το στοιχείο αυτό αποτελεί τον τελευταίο κόμβο της συνδεδεμένης λίστας*/

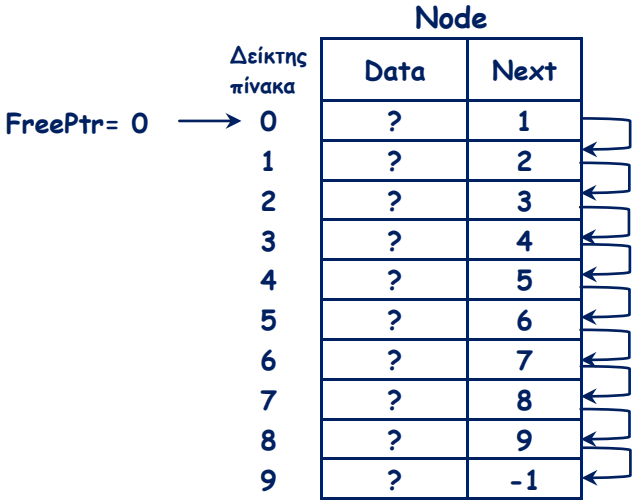
3. *FreePtr* ← 0

   /*Θέσε στο δείκτη *FreePtr* που δείχνει στον πρώτο διαθέσιμο κόμβο της συνδεδεμένης λίστας την τιμή 0, έτσι ώστε να δείχνει στον πρώτο κόμβο της συνδεδεμένης λίστας*/

παραπάνω, μένει, λοιπόν, να περιγράψουμε τον τρόπο οργάνωσης της δεξαμενής των 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 δείκτη (NilValue=-1). Τέλος, ένας δείκτης FreePtr θα έχει τιμή 0 για να δείχνει στον πρώτο κόμβο. Η διαδικασία αρχικοποίησης της δεξαμενής περιγράφεται με τον ακόλουθο αλγόριθμο:

Ο πίνακας Node θα είναι αρχικά:



Η κλίση 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.

GetNode

/*Δέχεται: Τον πίνακα *Node*, στον οποίο υπάρχουν - ενδεχομένως - αποθηκευμένα στοιχεία της συνδεδεμένης λίστας και ελεύθερες θέσεις που αποτελούν τη δεξαμενή των διαθέσιμων κόμβων, και τον δείκτη *FreePtr*.

Λειτουργία: Αποκτά έναν ελεύθερο κόμβο *TempPtr*.

Επιστρέφει: Τον δείκτη *TempPtr* και τον τροποποιημένο δείκτη *FreePtr* που δεικτοδοτεί σε διαθέσιμο κόμβο.*/

1. *TempPtr* ← *FreePtr* /*Θέσε στο δείκτη *TempPtr* την τιμή του δείκτη *FreePtr*, δηλαδή την τιμή του πρώτου διαθέσιμου κόμβου της συνδεδεμένης λίστας*/
2. **Αν** *FreePtr* != -1 **τότε** /*αν υπάρχουν διαθέσιμοι κόμβοι*/

   FreePtr ← *Node*[*FreePtr*].*Next*

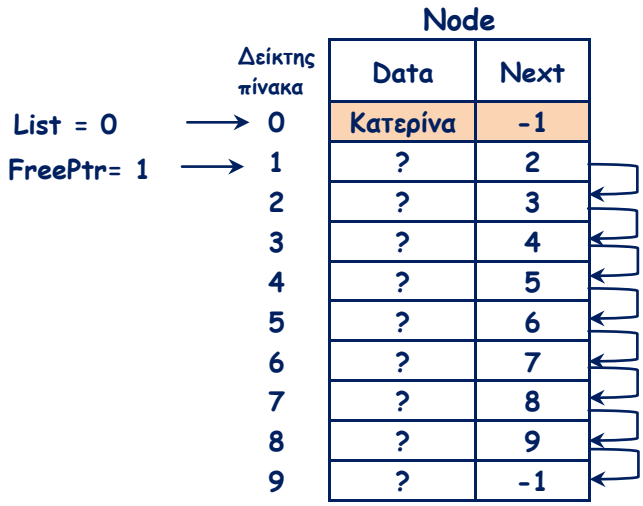
   /*Διαγράφεται ο πρώτος διαθέσιμος κόμβος από τη δεξαμενή των διαθέσιμων κόμβων, ενημερώνοντας το δείκτη *FreePtr* ώστε να δείχνει στον αμέσως επόμενο διαθέσιμο κόμβο*/

Αλλιώς

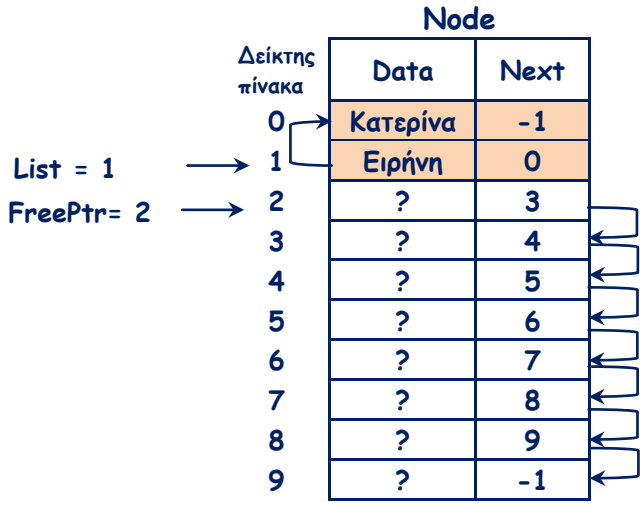
Γράψε 'Κενή λίστα'

Τέλος-αν

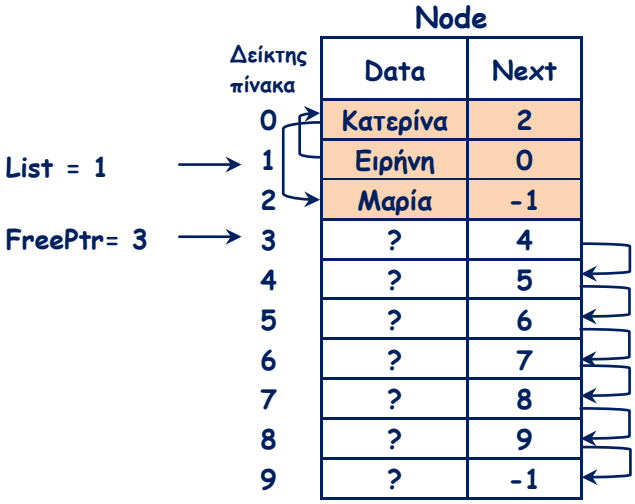
Έτσι, λοιπόν, αν το πρώτο στοιχείο που πρόκειται να εισαχθεί είναι το όνομα *Κατερίνα* θα αποθηκευτεί στην πρώτη θέση του πίνακα Node, αφού FreePtr=0. Η μεταβλητή List θα έχει τιμή 0 και η FreePtr θα πάρει τώρα την τιμή 1, όπως φαίνεται παρακάτω:



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:



Αν τώρα θέλουμε να εισαγάγουμε το όνομα *Μαρία*, τότε θα τοποθετηθεί στη θέση FreePtr=2, η FreePtr θα γίνει ίση με 3, η Node[0].Next θα πάρει την τιμή 2 και η Node[2].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.

ReleaseNode

/*Δέχεται: Τον πίνακα *Node*, που αναπαριστά τη δεξαμενή των διαθέσιμων κόμβων, και έναν δείκτη *TempPtr*.

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

Επιστρέφει: Τον τροποποιημένο πίνακα *Node*.*/

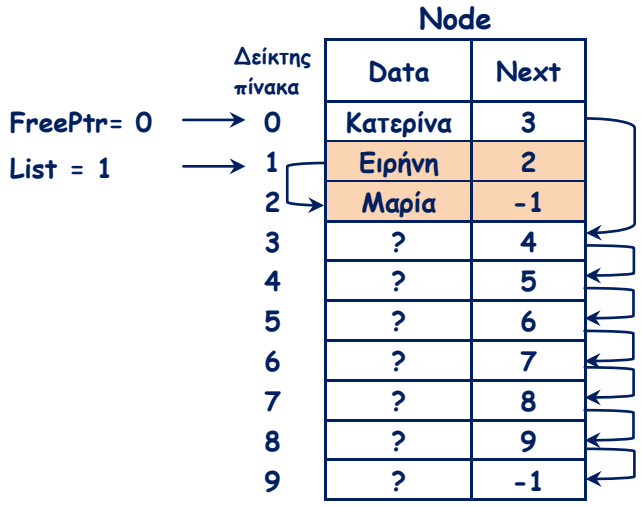
1. *Node[TempPtr].Next ← FreePtr*

/*Ενημέρωσε τον κόμβο που επιστρέφει στη δεξαμενή των διαθέσιμων κόμβων, δηλαδή το πεδίο *Next* της θέσης *TempPtr* του πίνακα *Node*, ώστε να δείχνει στην αρχή της συνδεδεμένης λίστας των διαθέσιμων κόμβων, δηλαδή στον κόμβο που δείχνει ο δείκτης *FreePtr* */

2. *FreePtr ← TempPtr*

/*Ενημέρωσε το δείκτη *FreePtr* που δεικτοδοτεί τον πρώτο κόμβο της λίστας των διαθέσιμων κόμβων ώστε να δείχνει στον κόμβο που επιστράφηκε στη δεξαμενή των διαθέσιμων κόμβων, δηλαδή στο κόμβο που δείχνει ο *TempPtr*/

Αν, για παράδειγμα, θέλουμε να διαγράψουμε το στοιχείο *Κατερίνα*, τότε θέτουμε Node[0].Next=FreePtr και FreePtr=0. Ο πίνακας Node είναι τώρα όπως φαίνεται παρακάτω:



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.

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

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

```
boolean EmptyList(ListPointer List)
/*Δέχεται:          Έναν δείκτη List που δείχνει σε μια συνδεδεμένη λίστα.
  Λειτουργία:       Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.
  Επιστρέφει:       TRUE αν η συνδεδεμένη λίστα είναι κενή και FALSE διαφορετικά.*/
{
  return (List==NilValue)
}
```

Σύμφωνα με τα παραπάνω, μπορεί να κατασκευαστεί ένα πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα, όπως το L_ListADT.c, και να χρησιμοποιηθεί σε ένα πρόγραμμα-πελάτη της C με την εντολή

#include "L_ListADT.h";

Ως ένα παράδειγμα χρήσης του ΑΤΔ Συνδεδεμένη Λίστα με πίνακα, κατασκευάστηκε το πρόγραμμα-πελάτης Reverse1.c, που εμφανίζει το ανάστροφο ενός συνόλου χαρακτήρων και χρησιμοποιεί τη διασύνδεση L_ListChADT.h (υλοποίηση L_ListChADT.c) για την υλοποίηση του ΑΤΔ Συνδεδεμένη Λίστα. Το πακέτο (διασύνδεσης και υλοποίησης) είναι ίδιο με το παραπάνω, με τη διαφορά ότι ο τύπος των στοιχείων ListElementType της λίστας είναι char και όχι int.

/*Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με πίνακα*/

```
// Filename L_ListADT.h

#define NumberOfNodes 50        /*όριο μεγέθους της συνδεδεμένης λίστας,
                                ενδεικτικά τέθηκε ίσο με 50.*/

#define NilValue -1             /*ειδική μηδενική τιμή*/

typedef int ListElementType;    /*τύπος δεδομένων για τα στοιχεία της
                                συνδεδεμένης λίστας, ενδεικτικά επιλέχθηκε ο
                                τύπος char*/

typedef int ListPointer;

typedef struct    {
                   ListElementType Data;
                   ListPointer Next;
                  } NodeType;
typedef enum {
               FALSE, TRUE
} boolean;

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);
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[]);

// Filename L_ListADT.c
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 προς το πρώτο
              διαθέσιμο κόμβο.*/
{
   int i;
    for (i=0; i< NumberOfNodes-1;i++)
   {
        Node[i].Next=i+1;
        Node[i].Data=-1;
    }
    Node[NumberOfNodes-1].Next= NilValue;
    Node[NumberOfNodes-1].Data= NilValue;
    *FreePtr=0;
}

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

boolean EmptyLList(ListPointer List)
/*Δέχεται:   Έναν δείκτη List που δείχνει σε μια συνδεδεμένη λίστα.
Λειτουργία:  Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.
Επιστρέφει:  TRUE αν η συνδεδεμένη λίστα είναι κενή και FALSE διαφορετικά.*/
{
  return (List==NilValue);
}

boolean FullLList(ListPointer FreePtr)
/*Δέχεται:   Μια συνδεδεμένη λίστα.
Λειτουργία:  Ελέγχει αν η συνδεδεμένη λίστα είναι γεμάτη.
Επιστρέφει:  TRUE αν η συνδεδεμένη λίστα είναι γεμάτη, FALSE διαφορετικά.*/
{
  return (FreePtr==NilValue);
}

void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[])
/*Δέχεται:    Τον πίνακα Node και τον δείκτη FreePtr.
Λειτουργία:   Αποκτά έναν "ελεύθερο" κόμβο.
Επιστρέφει:   Τον δείκτη P και τον τροποποιημένο  δείκτη FreePtr που δεικτοδοτεί σε
              διαθέσιμο κόμβο.*/
{
  *P = *FreePtr;
  if (!FullLList(*FreePtr))
    *FreePtr =Node[*FreePtr].Next;
}
```

```
void ReleaseNode(NodeType Node[], ListPointer P, ListPointer *FreePtr)
/*Δέχεται:    Τον πίνακα Node, που αναπαριστά τη δεξαμενή των διαθέσιμων κόμβων,
              έναν δείκτη P και το δείκτη FreePtr.
Λειτουργία:  Επιστρέφει στη δεξαμενή τον κόμβο στον οποίο δείχνει ο P
Επιστρέφει:  Τον τροποποιημένο πίνακα Node και τον δείκτη FreePtr.*/
{
  Node[P].Next =*FreePtr;
  Node[P].Data = -1;     /*Όχι αναγκαία εντολή, βοηθητική για να φαίνονται στην
                           εμφάνιση οι διαγραμμένοι κόμβοι */
  *FreePtr =P;
}

 void Insert(ListPointer *List, NodeType Node[],ListPointer
            *FreePtr,ListPointer PredPtr, ListElementType Item)
/*Δέχεται:     Μια συνδεδεμένη λίστα, τον πίνακα Node, 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, τον τροποποιημένο πίνακα Node
               και τον δείκτη FreePtr.
Έξοδος:      Μήνυμα γεμάτης λίστας, αν η συνδεδεμένη λίστα είναι γεμάτη.*/
{
  ListPointer TempPtr;
  GetNode(&TempPtr,FreePtr,Node);
  if (!FullLList(TempPtr))
  {
     if (PredPtr==NilValue)
    {
        Node[TempPtr].Data =Item;
        Node[TempPtr].Next =*List;
        *List =TempPtr;
    }
    else
    {
        Node[TempPtr].Data =Item;
        Node[TempPtr].Next =Node[PredPtr].Next;
        Node[PredPtr].Next =TempPtr;
    }
  }
  else
    printf("Full List ...\n");
}

 void Delete(ListPointer *List, NodeType Node[], ListPointer *FreePtr,
ListPointer PredPtr)
/*Δέχεται:    Μια συνδεδεμένη λίστα 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 προηγούμενο
              κόμβο από αυτόν στον οποίο δείχνει ο PredPtr.
```

```
Επιστρέφει:   Την τροποποιημένη λίστα και το δείκτη FreePtr.
Έξοδος:       Μήνυμα κενής λίστας, αν η συνδεδεμένη λίστα είναι κενή.*/
{
  ListPointer TempPtr ;

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

void TraverseLinked(ListPointer List, NodeType Node[])
/*Δέχεται:      Μια συνδεδεμένη λίστα.
Λειτουργία:   Κάνει διάσχιση της συνδεδεμένης λίστας, αν δεν είναι κενή.
Έξοδος:       Εξαρτάται από την επεξεργασία.*/
{
  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;
       }
       printf("\n");
   }
  else printf("Empty List ...\n");
}
```