4.6 Ένα πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα

Στην ενότητα 4.2 περιγράφονται οι διαδικασίες εισαγωγής στοιχείου σε μια λίστα και διαγραφής ενός στοιχείου από αυτήν, όμως δεν δίνονται και οι αντίστοιχοι αλγόριθμοι.

Για την εισαγωγή ενός στοιχείου, είπαμε ότι πρέπει πρώτα να αποκτήσουμε έναν νέο κόμβο και έπειτα να τον συνδέσουμε με την λίστα. Μάλιστα διακρίναμε δύο περιπτώσεις: ο νέος κόμβος να εισαχθεί α) στην αρχή της λίστας και β) μετά από κάποιον συγκεκριμένο κόμβο της λίστας. Ο αλγόριθμος για τη λειτουργία της εισαγωγής σε μια συνδεδεμένη λίστα υλοποιημένη με δείκτες είναι ο ακόλουθος:

LinkedInsert

/*Αλγόριθμος εισαγωγής στοιχείου σε συνδεδεμένη λίστα.

Δέχεται: Μια συνδεδεμένη λίστα με τον *List* να δείχνει στον πρώτο κόμβο, ένα στοιχείο δεδομένων *Item* και έναν δείκτη *PredPtr*.

Λειτουργία: Εισάγει έναν κόμβο, που περιέχει το *Item,* στην συνδεδεμένη λίστα μετά από τον κόμβο που δεικτοδοτείται από τον *PredPtr* ή στην αρχή της λίστας, αν ο *PredPtr* είναι μηδενικός (NULL).

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

1. Πάρε έναν κόμβο δεικτοδοτούμενο από τον *TempPtr*
2. Data(*TempPtr*) ← *Item*

   /*Θέσε στο πεδίο *Data* του κόμβου που δείχνει ο *TempPtr* την τιμή της *Item**/
3. **Αν** *PredPtr* = NULL **τότε**

   /*αν το στοιχείο *Item* δεν έχει προηγούμενο κόμβο*/

   /*Εισήγαγε το *Item* στην αρχή της λίστας*/

   α. Next(*TempPtr*) ← *List*

   /*Θέσε στο πεδίο *Next* του κόμβου που δείχνει ο *TempPtr* την τιμή της *List*, δηλαδή επόμενος κόμβος του *TempPtr* είναι ο *List**/

   β. *List* ← *TempPtr*

   /*Θέσε στο δείκτη *List* την τιμή του δείκτη *TempPtr*, δηλαδή θέσε ως νέο πρώτο κόμβο της συνδεδεμένης λίστας τον κόμβο εκείνο που έδειχνε μέχρι τώρα ο δείκτης *TempPtr**/

Αλλιώς

/*υπάρχει προηγούμενο στοιχείο*/

α. Next(*TempPtr*) ← Next(*PredPtr*)

/*Θέσε στο πεδίο *Next* του κόμβου του δείκτη *TempPtr* την τιμή του πεδίου *Next* του κόμβου *PredPtr*, δηλαδή επόμενος κόμβος του *TempPtr* είναι ο κόμβος που είναι επόμενος του */

β. Next(*PredPtr*) ← *TempPtr*

/*Θέσε στο πεδίο *Next* του δείκτη *PredPtr* την τιμή του δείκτη *TempPtr* , δηλαδή επόμενος κόμβος του *PredPtr* είναι ο *TempPtr* */

Τέλος_αν

Για τη διαγραφή ενός στοιχείου από τη λίστα υπάρχουν πάλι δύο περιπτώσεις: α) διαγραφή του πρώτου στοιχείου της λίστας και β) διαγραφή ενός στοιχείου που έχει προηγούμενο. Ένας αλγόριθμος για τη λειτουργία της διαγραφής ενός στοιχείου από μια συνδεδεμένη λίστα υλοποιημένη με δείκτες είναι ο ακόλουθος:

LinkedDelete

/*Αλγόριθμος διαγραφής στοιχείου από συνδεδεμένη λίστα

Δέχεται:	Μια συνδεδεμένη λίστα με τον *List* να δείχνει στον πρώτο κόμβο της και έναν δείκτη *PredPtr*.
Λειτουργία:	Διαγράφει από τη συνδεδεμένη λίστα τον κόμβο που έχει για προηγούμενό του αυτόν στον οποίο δείχνει ο *PredPtr* ή διαγράφει τον πρώτο κόμβο, αν ο *PredPtr* είναι μηδενικός, εκτός και αν η λίστα 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 ήταν κενή*/

Αν η λίστα είναι κενή **τότε**

Γράψε 'Η λίστα είναι κενή'

Αλλιώς

1. **Αν** *PredPtr* = NULL **τότε**

/*διαγραφή του πρώτου στοιχείου*/

α. *TempPtr* ← *List*

/*θέσε στον *TempPtr* την τιμή της *List*, δηλαδή ο δείκτης *TempPtr* θα δείχνει στον κόμβο που δείχνει και ο δείκτης *List**/

β. *List* ← *Next*(*TempPtr*)

/*θέσε στη *List* την τιμή του πεδίου *Next* του δείκτη του κόμβου *TempPtr*, δηλαδή ο πρώτος κόμβος της συνδεδεμένης λίστας θα είναι αυτός που ήταν επόμενος του κόμβου που έδειχνε ο *TempPtr*, δηλαδή ο μέχρι τώρα 2ος κόμβος*/

Αλλιώς

/*διαγραφή στοιχείου που έχει προηγούμενο*/

α. *TempPtr* ← *Next*(*PredPtr*)

/*θέσε στο δείκτη *TempPtr* την τιμή του πεδίου *Next* του κόμβου που δείχνει ο *PredPtr*, δηλαδή ο δείκτης *TempPtr* δείχνει τον κόμβο που είναι επόμενος του κόμβου που δείχνει ο *PredPtr**/

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

/*θέσε στο πεδίο *Next* του κόμβου του δείκτη *PredPtr* την τιμή του πεδίου *Next* του κόμβου του δείκτη *TempPtr,* δηλαδή επόμενος κόμβος του *PredPtr* είναι ο κόμβος που είναι επόμενος του *TempPtr**/

/*Δηλαδή μ' αυτές τις 2 παραπάνω εντολές παρακάμπτω τον επόμενο κόμβο του *PredPtr*, και κάνω επόμενο του *PredPtr,* τον μεθεπόμενό του*/

Τέλος_αν

2.Να επιστρέψεις στην δεξαμενή των διαθέσιμων κόμβων τον κόμβο στον οποίο δείχνει ο *TempPtr*

Τέλος_αν

Αν η σειρά των στοιχείων της λίστας δεν μας ενδιαφέρει, τότε δεν έχει σημασία πού θα εισαχθεί ένα νέο στοιχείο κι επομένως οι εισαγωγές μπορούν να γίνονται πάντα στην αρχή της λίστας. Σε αυτήν την περίπτωση πρόκειται για μη ταξινομημένη λίστα και το τρίτο βήμα στον αλγόριθμο εισαγωγής γίνεται απλά:

3. Θέσε Next(*TempPtr*) <- *List*.
4. Θέσε *List* <- *TempPtr*.

Για τη διαγραφή στοιχείου χρειάζεται να τοποθετήσουμε το δείκτη PredPtr δοσμένης της τιμής του στοιχείου που θα διαγραφεί. Επομένως, πρέπει να αναζητήσουμε μέσα στη λίστα για να βρούμε τη θέση αυτού του στοιχείου. Ένας αλγόριθμος αναζήτησης συνδεδεμένης λίστας είναι ο ακόλουθος:

LinearSearch

/*Αλγόριθμος αναζήτησης σε συνδεδεμένη λίστα.

Δέχεται: Ένα στοιχείο *Item* και μια συνδεδεμένη λίστα στην οποία ο δείκτης *List* δείχνει στον πρώτο κόμβο.

Λειτουργία: Εκτελεί γραμμική αναζήτηση της συνδεδεμένης λίστας για έναν κόμβο που να περιέχει το στοιχείο *Item*.

Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η *Found* είναι αληθής, ο *CurrPtr* δείχνει στον κόμβο που περιέχει το *Item* και ο *PredPtr* στον προηγούμενό του ή είναι NULL αν δεν υπάρχει προηγούμενος. Αν η αναζήτηση δεν είναι επιτυχής η *Found* είναι ψευδής.*/

1. *CurrPtr* ← *List*

/*θέσε στον *CurrPtr* την τιμή του δείκτη *List*, δηλαδή ο *CurrPtr* δείχνει τον 1ο κόμβο της συνδεδεμένης λίστας*/

PredPtr ← NULL

/*Θέσε στο *PredPtr* την τιμή NULL*/

/*οι δείκτες *CurrPtr* και *PredPtr* θα μετακινούνται ταυτόχρονα ώστε να δείχνουν αντίστοιχα τον τρέχοντα κόμβο και τον προηγούμενό του*/

Found ← **FALSE**

2. **Όσο** *Found* = **FALSE και** *CurrPtr* != NULL **επανάλαβε**

/*όσο δεν βρέθηκε το στοιχείο και δεν διέσχισες όλη τη συνδεδεμένη λίστα*/

Αν *Data*(*CurrPtr*) =*Item* **τότε**

/*αν η τιμή του πεδίου *Data* του κόμβου που δείχνει ο *CurrPtr* είναι ίση με την τιμή του *Item**/

Found ← **TRUE**

/*το στοιχείο βρέθηκε*/

Αλλιώς

α. *PredPtr* ← *CurrPtr*

/*μετακίνησε το δείκτη *PredPtr* ώστε να δείχνει στον κόμβο που έδειχνε ο δείκτης *CurrPtr**/

β. *CurrPtr* ← *Next*(*CurrPtr*)

/*μετακίνησε το δείκτη *CurrPtr* ώστε να δείχνει στο μέχρι τώρα κόμβο που ήταν επόμενος του κόμβου που έδειχνε ο δείκτης *CurrPtr**/

Τέλος_αν

Τέλος_επανάληψης

Σε μια ταξινομημένη λίστα οι κόμβοι είναι συνδεδεμένοι μεταξύ τους με τρόπο ώστε να επισκεπτόμαστε τα στοιχεία που είναι αποθηκευμένα στους κόμβους κατά αύξουσα ή φθίνουσα σειρά καθώς διασχίζουμε τη λίστα. Αν το τμήμα δεδομένων ενός κόμβου είναι εγγραφή, τότε ένα από τα πεδία της εγγραφής αποτελεί το πεδίο **κλειδί** (**key**) και η ταξινόμηση γίνεται με βάση την τιμή που έχει το πεδίο αυτό. Σε κάθε εισαγωγή νέου στοιχείου πρέπει τα στοιχεία να παραμένουν ταξινομημένα. Ο αλγόριθμος αναζήτησης για μια ταξινομημένη συνδεδεμένη λίστα είναι ο ακόλουθος:

OrderedLinearSearch

/*Αλγόριθμος αναζήτησης σε ταξινομημένη συνδεδεμένη λίστα.

Δέχεται: Ένα στοιχείο *Item* και μια ταξινομημένη συνδεδεμένη λίστα, που περιέχει στοιχεία δεδομένων σε αύξουσα διάταξη και στην οποία ο δείκτης *List* δείχνει στον πρώτο κόμβο.

Λειτουργία: Εκτελεί γραμμική αναζήτηση της συνδεδεμένης λίστας για τον πρώτο κόμβο που περιέχει το στοιχείο *Item* ή για μια θέση για να εισάγει ένα νέο κόμβο που να περιέχει το στοιχείο *Item*.

Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η *Found* είναι αληθής, ο *CurrPtr* δείχνει στον κόμβο που περιέχει το *Item* και ο *PredPtr* στον προηγούμενό του ή είναι NULL αν δεν υπάρχει προηγούμενος. Αν η αναζήτηση δεν είναι επιτυχής η *Found* είναι ψευδής.*/

1. *CurrPtr* ← *List*

/*θέσε στον *CurrPtr* την τιμή του δείκτη *List*, δηλαδή ο *CurrPtr* δείχνει τον 1ο κόμβο της ταξινομημένης συνδεδεμένης λίστας*/

PredPtr ← NULL

/*Θέσε στο *PredPtr* την τιμή NULL*/

/*οι δείκτες *CurrPtr* και *PredPtr* θα μετακινούνται ταυτόχρονα ώστε να δείχνουν αντίστοιχα τον τρέχοντα κόμβο και τον προηγούμενό του*/

Found ← **FALSE**

DoneSearching ← **FALSE**

2. **Όσο** *DoneSearching* = **FALSE και** *CurrPtr* != NULL **επανάλαβε**

/*όσο η αναζήτηση δεν τελείωσε, δηλαδή όσο το στοιχείο που ελέγχθηκε στην αμέσως προηγούμενη επανάληψη είναι μικρότερο από το *Item*, και δεν διέσχισες όλη την ταξινομημένη συνδεδεμένη λίστα*/

Αν *Data*(*CurrPtr*) ≥ *Item* **τότε**

```
                                    /*η τιμή Data του τρέχοντα κόμβου που δεικτοδοτείται από
                                    τον CurrPtr είναι μεγαλύτερη ή ίση του στοιχείου Item που
                                    αναζητείται*/
    α. DoneSearching ← TRUE

                                    /*η αναζήτηση τελείωσε*/
    β. Αν Data(CurrPtr) = Item τότε

                                    /*η τιμή Data του τρέχοντα κόμβου που δεικτοδοτείται από
                                    τον CurrPtr είναι ίση με το στοιχείο Item που αναζητείται*/
       Found ← TRUE

                                    /*το στοιχείο βρέθηκε*/
    Τέλος_αν
  Αλλιώς
    α. PredPtr ← CurrPtr

                                    /*μετακίνησε το δείκτη PredPtr ώστε να δείχνει στον κόμβο
                                    που έδειχνε ο δείκτης CurrPtr*/
    β. CurrPtr ← Next(CurrPtr)

                                    /*μετακίνησε το δείκτη CurrPtr ώστε να δείχνει στο μέχρι
                                    τώρα κόμβο που ήταν επόμενος του κόμβου που έδειχνε ο
                                    δείκτης CurrPtr*/
  Τέλος_αν
Τέλος_επανάληψης
```

Αφού περιγράψαμε πώς υλοποιούνται όλες οι λειτουργίες των συνδεδεμένων λιστών με δείκτες, μπορούμε τώρα να δούμε συνολικά ένα πακέτο για τις συνδεδεμένες λίστες σε C. Παρακάτω φαίνεται η διασύνδεση ListPADT.h και η υλοποίηση της ListPADT.c, που μπορεί να χρησιμοποιηθεί σε προγράμματα-πελάτες της C με την εντολή

```
#include "ListPADT.h";
```

```
/*Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες*/

// Filename ListPADT.h

typedef int ListElementType;                    /*ο τύπος των στοιχείων της
                                                συνδεδεμένης λίστας*/
typedef struct ListNode *ListPointer;           /*ο τύπος των δεικτών για τους
                                                κόμβους*/
typedefstruct ListNode {
                    ListElementType Data;
                    ListPointer Next;
             } ListNode;
 typedef enum {
             FALSE, TRUE
} boolean;
void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer
PredPtr);
void LinkedDelete(ListPointer *List, ListPointer PredPtr);
void LinkedTraverse(ListPointer List);
void LinearSearch(ListPointer List, ListElementType Item, ListPointer
*PredPtr, boolean *Found);
void OrderedLimearSearch(ListPointer List, ListElementType Item, ListPointer
*PredPtr, boolean *Found);

// Filename ListPADT.c

#include <stdio.h>
#include <stdlib.h>
#include "ListPADT.h"

 void CreateList(ListPointer *List)
 /*Λειτουργία:          Δημιουργεί μια κενή συνδεδεμένη λίστα.
   Επιστρέφει:          Τον μηδενικό δείκτη List.*/
{
     *List = NULL;
}

  boolean EmptyList(ListPointer List)
 /*Δέχεται:             Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο
                       κόμβο.
   Λειτουργία:          Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.
   Επιστρέφει:          TRUE αν η λίστα είναι κενή και FALSE διαφορετικά.*/
{
     return (List==NULL);
}
```

```
void LinkedTraverse(ListPointer List)
/*Δέχεται:            Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
  Λειτουργία:        Διασχίζει τη συνδεδεμένη λίστα και επεξεργάζεται κάθε δεδομένο
                     ακριβώς μια φορά.
  Επιστρέφει:        Εξαρτάται από το είδος της επεξεργασίας.*/
{
ListPointer CurrPtr;          /*δείκτης για τον τρέχοντα κόμβο που
                                επεξεργάζεται*/
   if (EmptyList(List))
       printf("EMPTY LIST\n");
   else
   {
     CurrPtr = List;
         while (CurrPtr!=NULL)
        {
           printf("%d\n", CurrPtr-> Data);
            CurrPtr = CurrPtr-> Next;
        }
   }
}

 void LinearSearch(ListPointer List, ListElementType Item, ListPointer
                   *PredPtr, boolean *Found)
/*Δέχεται:            Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
  Λειτουργία:        Εκτελεί μια γραμμική αναζήτηση στην μη ταξινομημένη
                     συνδεδεμένη λίστα για έναν κόμβο που να περιέχει το στοιχείο
                     Item.
  Επιστρέφει:        Αν η αναζήτηση είναι επιτυχής η Found είναι TRUE, ο CurrPtr
                     δείχνει στον κόμβο που περιέχει το Item και ο PredPtr στον
                     προηγούμενό του ή είναι NULL αν δεν υπάρχει προηγούμενος. Αν η
                     αναζήτηση δεν είναι επιτυχής η Found είναι FALSE.*/
{
    ListPointer CurrPtr;                    /*ο δείκτης για τον τρέχοντα κόμβο*/
    boolean stop;
   CurrPtr = List;
  *PredPtr = NULL;
  stop = FALSE;
  while (!stop && CurrPtr!=NULL )
   {
        if (CurrPtr->Data==Item )
                stop = TRUE;
        else
        {
           *PredPtr= CurrPtr;
           CurrPtr = CurrPtr->Next;
        }
       }
       *Found=stop;
}
```

```
void OrderedLinearSearch(ListPointer List, ListElementType Item, ListPointer
                         *PredPtr, boolean *Found)
/*Δέχεται:        Ένα στοιχείο Item και μια ταξινομημένη συνδεδεμένη λίστα, που
                  περιέχει στοιχεία δεδομένων σε αύξουσα διάταξη και στην οποία
                  ο δείκτης List δείχνει στον πρώτο κόμβο.
  Λειτουργία:     Εκτελεί γραμμική αναζήτηση της συνδεδεμένης ταξινομημένης
                  λίστας για τον πρώτο κόμβο που περιέχει το στοιχείο Item ή
                  για μια θέση για να εισάγει ένα νέο κόμβο που να περιέχει το
                  στοιχείο Item.
  Επιστρέφει:     Αν η αναζήτηση είναι επιτυχής η Found είναι TRUE, ο CurrPtr
                  δείχνει στον κόμβο που περιέχει το Item και ο PredPtr στον
                  προηγούμενό του ή είναι NULL αν δεν υπάρχει προηγούμενος. Αν
                  η αναζήτηση δεν είναι επιτυχής η Found είναι FALSE.*/
{
   ListPointer CurrPtr;             /*ο δείκτης για τον τρέχοντα κόμβο*/
   boolean DoneSearching;           /*σημαίνει το τέλος της αναζήτησης*/
   CurrPtr = List;
   *PredPtr = NULL;
   DoneSearching = FALSE;
   *Found = FALSE;
   while (!DoneSearching && CurrPtr!=NULL)
   {
         if (CurrPtr->Data>=Item )
         {
                 DoneSearching = TRUE;
                 *Found = (CurrPtr->Data==Item);
         }
         else
         {
               *PredPtr= CurrPtr;
               CurrPtr = CurrPtr->Next;
         }
   }
}

void LinkedInsert(ListPointer *List, ListElementType Item,
                  ListPointer PredPtr)
/*Δέχεται:        Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο
                  κόμβο, ένα στοιχείο δεδομένων Item και έναν δείκτη PredPtr.
  Λειτουργία:     Εισάγει έναν κόμβο, που περιέχει το Item, στην συνδεδεμένη
                  λίστα μετά από τον κόμβο που δεικτοδοτείται από τον PredPtr ή
                  στην αρχή της συνδεδεμένης λίστας, αν ο PredPtr είναι
                  μηδενικός(NULL).
 Επιστρέφει:      Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο της να
                  δεικτοδοτείται από τον List.*/
{
   ListPointer TempPtr;          /*δείχνει στο νέο κόμβο που θα εισαχθεί*/

   TempPtr= (ListPointer)malloc(sizeof(struct ListNode));
```

```
   TempPtr->Data = Item;
        if (PredPtr==NULL)
       {
              TempPtr->Next = *List;
              *List = TempPtr;
       }
       else {
             TempPtr->Next = PredPtr->Next;
             PredPtr->Next = TempPtr;
       }
}

void LinkedDelete(ListPointer *List, ListPointer PredPtr)
/*Δέχεται:          Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο
                    της και έναν δείκτη PredPtr.
  Λειτουργία:       Διαγράφει από τη συνδεδεμένη λίστα τον κόμβο που έχει για
                    προηγούμενό του αυτόν στον οποίο δείχνει ο PredPtr ή
                    διαγράφει τον πρώτο κόμβο, αν ο PredPtr είναι μηδενικός,
                    εκτός και αν η λίστα 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 ήταν κενή*/
{
   ListPointer TempPtr;              /*δείχνει στον κόμβο που θα διαγραφεί*/
    if (EmptyList(*List))
            printf("EMPTY LIST\n");
   else
   {
         if (PredPtr==NULL)
        {
              TempPtr = *List;
              *List = TempPtr->Next;
        }
        else
        {
             TempPtr = PredPtr->Next;
             PredPtr->Next = TempPtr->Next;
        }
        free(TempPtr);
    }
}
```

Στην ενότητα 4.3 υλοποιήσαμε τη συνδεδεμένη λίστα με πίνακα και χρησιμοποιήσαμε αυτόν τον ΑΤΔ για το πρόγραμμα-πελάτης Reverse1.c, που εμφανίζει το ανάστροφο ενός συνόλου χαρακτήρων. Το αποτέλεσμα θα είναι το ίδιο, αν υλοποιήσουμε τη συνδεδεμένη λίστα με δείκτες. Το πρόγραμμα-πελάτης Reverse2.c χρησιμοποιεί τη διασύνδεση LPChADT.h (υλοποίηση LPChADT.c) για να εμφανίσει ένα σύνολο χαρακτήρων

αντιστραμμένο. Η μόνη διαφορά του από το ListPADT.c είναι ότι τα στοιχεία είναι τύπου char και όχι int.