ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΛΙΣΤΕΣ
4.6 Ένα πακέτο για τον ΑΤΔ
Συνδεδεμένη Λίστα

Αλγόριθμος εισαγωγής στοιχείου

Για την εισαγωγή ενός στοιχείου, πρέπει πρώτα να αποκτήσουμε έναν νέο κόμβο και έπειτα να τον συνδέσουμε με την λίστα.

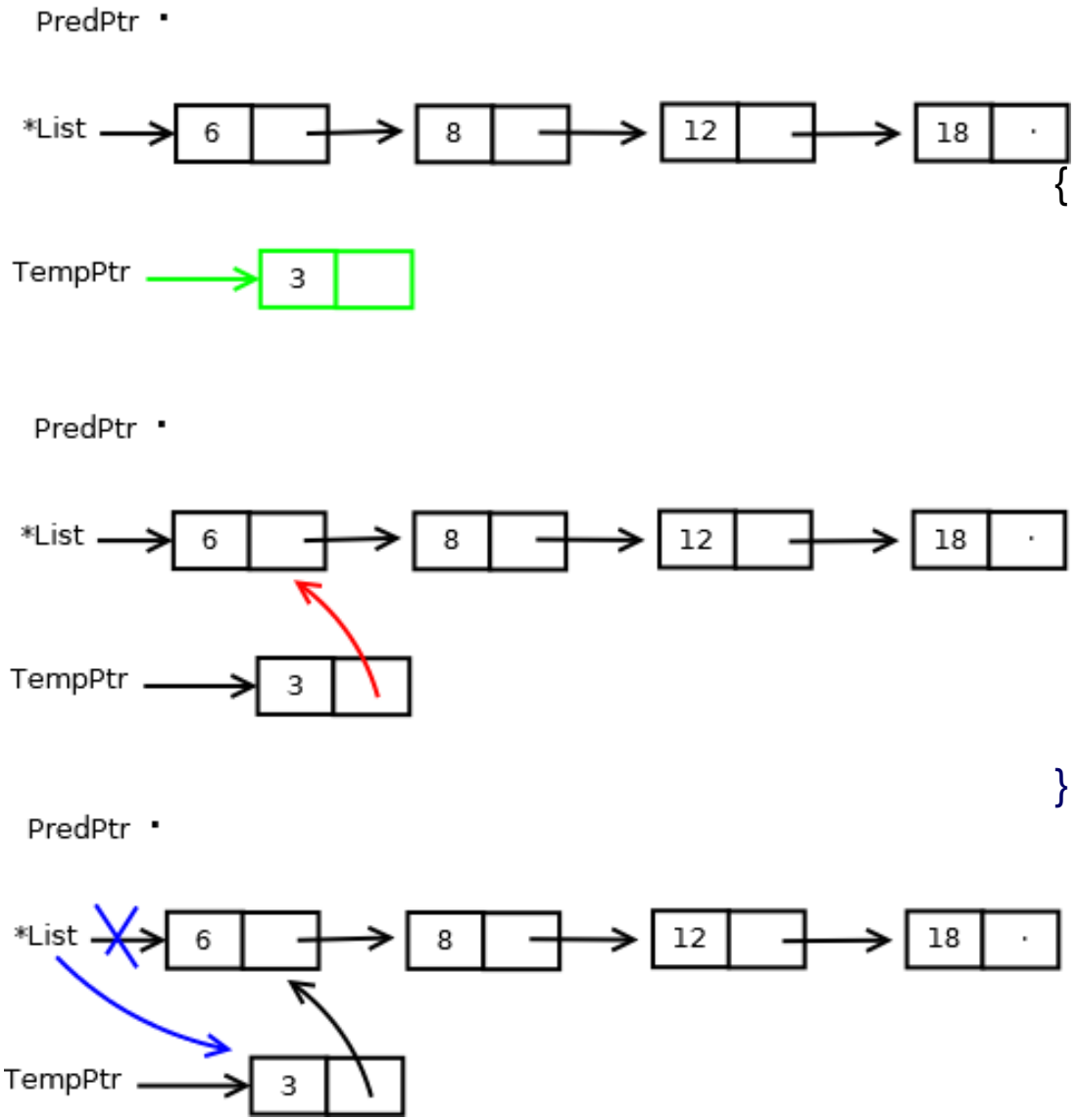
Υπάρχουν δύο περιπτώσεις: ο νέος κόμβος να εισαχθεί

α) στην αρχή της λίστας και

β) μετά από κάποιον συγκεκριμένο κόμβο της λίστας.

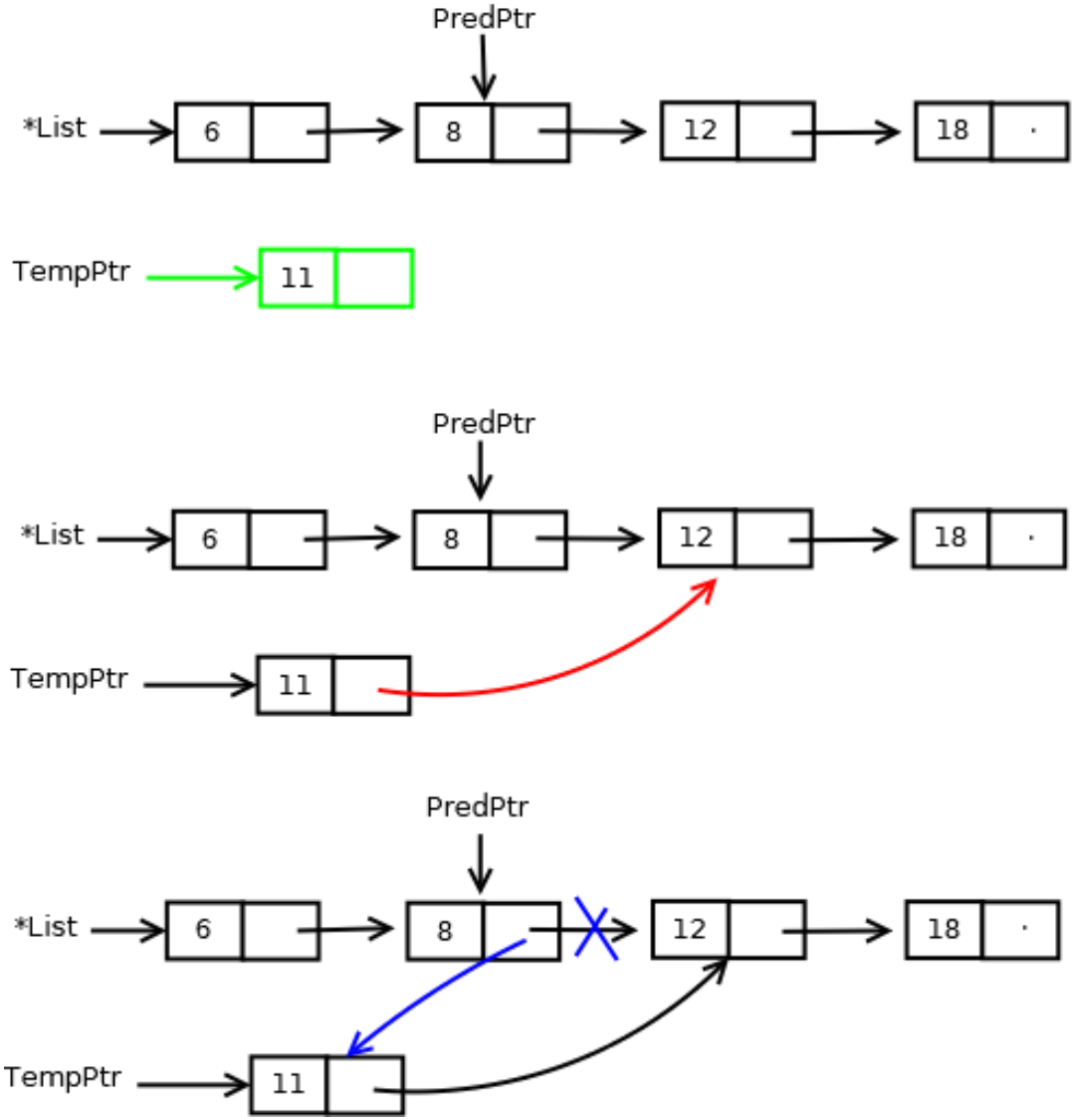
Ο αλγόριθμος για τη λειτουργία της εισαγωγής σε μια συνδεδεμένη λίστα υλοποιημένη με δείκτες είναι ο ακόλουθος:

Εισαγωγή στην αρχή της λίστας




```
void LinkedInsert(ListPointer *List,
                  ListElementType Item,
                  ListPointer PredPtr)
{
   ListPointer TempPtr;
   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;
   }
}
```

Εισαγωγή μετά από τον κόμβο PredPtr




```
void LinkedInsert(ListPointer *List,
                  ListElementType Item,
                  ListPointer PredPtr)
{
        ListPointer TempPtr;
    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;
    }
}
```

Εισαγωγή στοιχείου (LinkedInsert)

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*.*/

Εισαγωγή στοιχείου (LinkedInsert)

```
void LinkedInsert(ListPointer *List, ListElementType Item,
                  ListPointer PredPtr)
{
    ListPointer TempPtr;

  /*απόκτηση νέου κόμβου*/
   TempPtr = (ListPointer)malloc(sizeof(struct ListNode));
   TempPtr->Data = Item;

    if (PredPtr == NULL) {          /*εισαγωγή στην αρχή της λίστας*/
          TempPtr->Next = *List;
          *List = TempPtr;
   }
   else {     /*εισαγωγή μετά από κάποιον συγκεκριμένο κόμβο της λίστας(PredPtr)*/
                TempPtr->Next = PredPtr->Next;
                PredPtr->Next = TempPtr;
   }
}
```

Αλγόριθμος διαγραφής στοιχείου

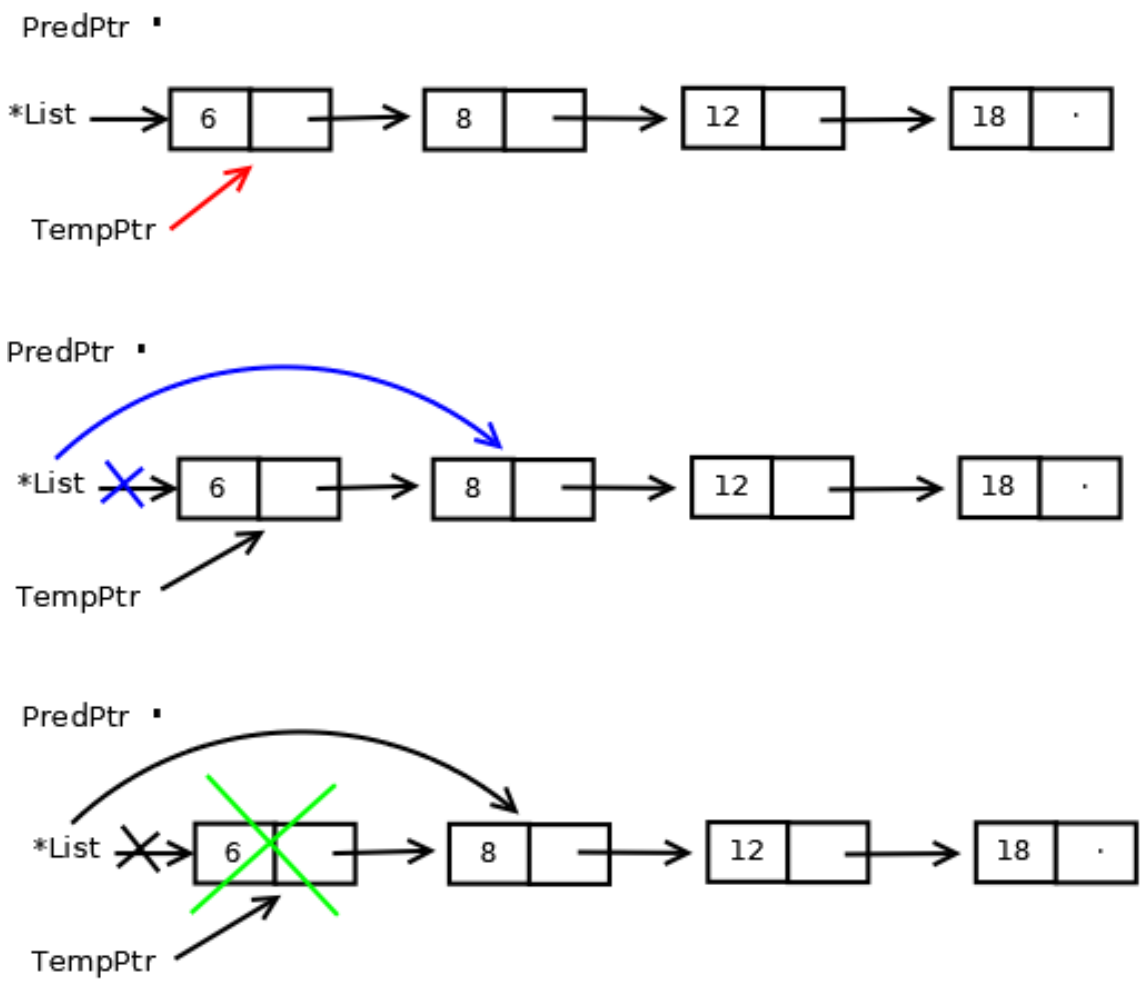
Για τη διαγραφή ενός στοιχείου από τη λίστα υπάρχουν πάλι δύο περιπτώσεις:

α) διαγραφή του πρώτου στοιχείου της λίστας και

β) διαγραφή ενός στοιχείου που έχει προηγούμενο.

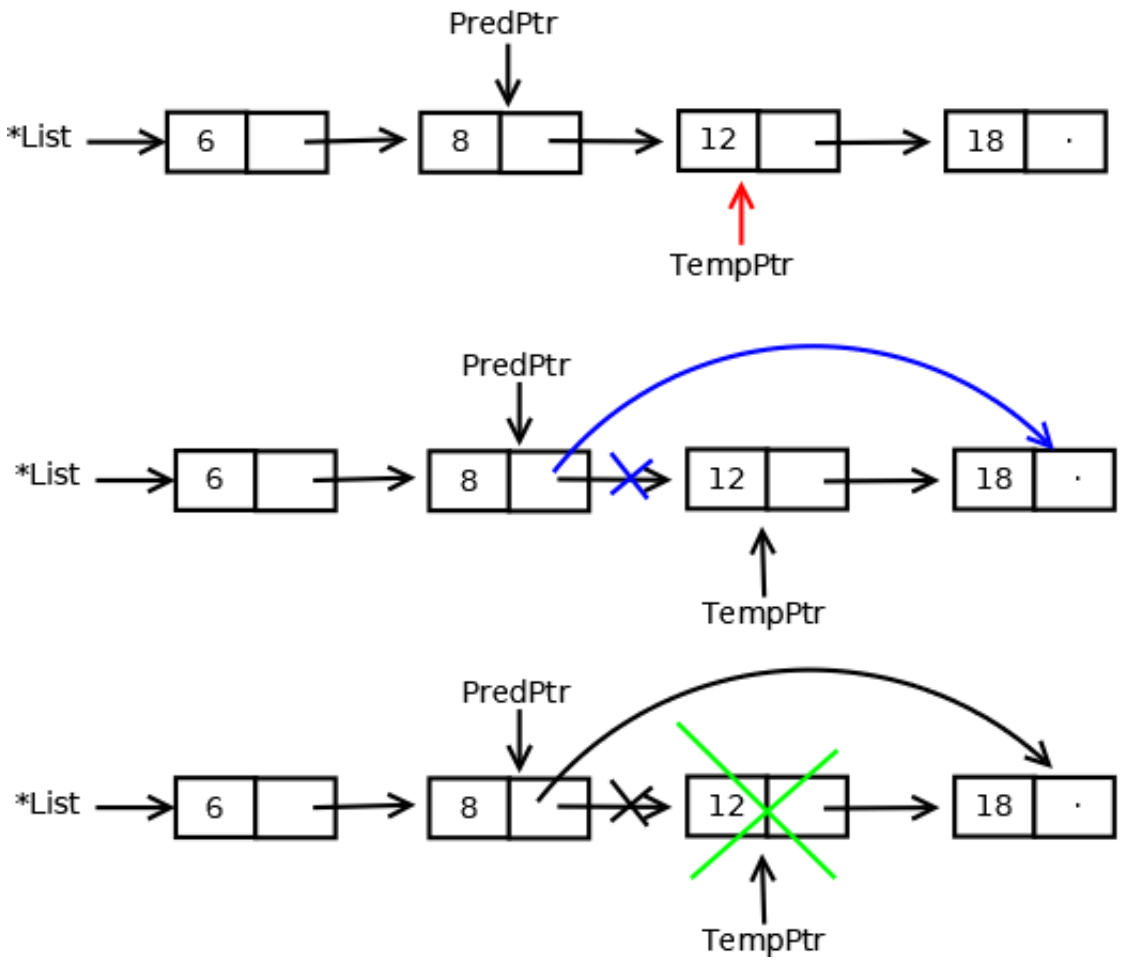
Ένας αλγόριθμος για τη λειτουργία της διαγραφής ενός στοιχείου από μια συνδεδεμένη λίστα υλοποιημένη με δείκτες είναι ο ακόλουθος:

Διαγραφή του πρώτου κόμβου της λίστας




```
void LinkedDelete(ListPointer *List,
     ListPointer PredPtr)
{
   ListPointer TempPtr;
   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);
  }
}
```

Διαγραφή άλλου κόμβου της λίστας




```
void LinkedDelete(ListPointer *List,
     ListPointer PredPtr)
{
   ListPointer TempPtr;
   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);
  }
}
```

Διαγραφή στοιχείου (LinkedDelete)

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*/

Διαγραφή στοιχείου (LinkedDelete)

```
void LinkedDelete(ListPointer *List, ListPointer PredPtr)
{
    ListPointer TempPtr;
    if (EmptyList(*List))    printf("EMPTY LIST\n");
   else {
        if (PredPtr == NULL) {   /*διαγραφή του πρώτου κόμβου της λίστας*/
                        TempPtr = *List;
                        *List = TempPtr->Next;
        }
        else {          /*διαγραφή ενός στοιχείου που έχει προηγούμενο*/
                        TempPtr = PredPtr->Next;
                        PredPtr->Next = TempPtr->Next;
        }
        free(TempPtr);
    }
}
```

Εισαγωγή/διαγραφή σε μη ταξινομημένη λίστα

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.

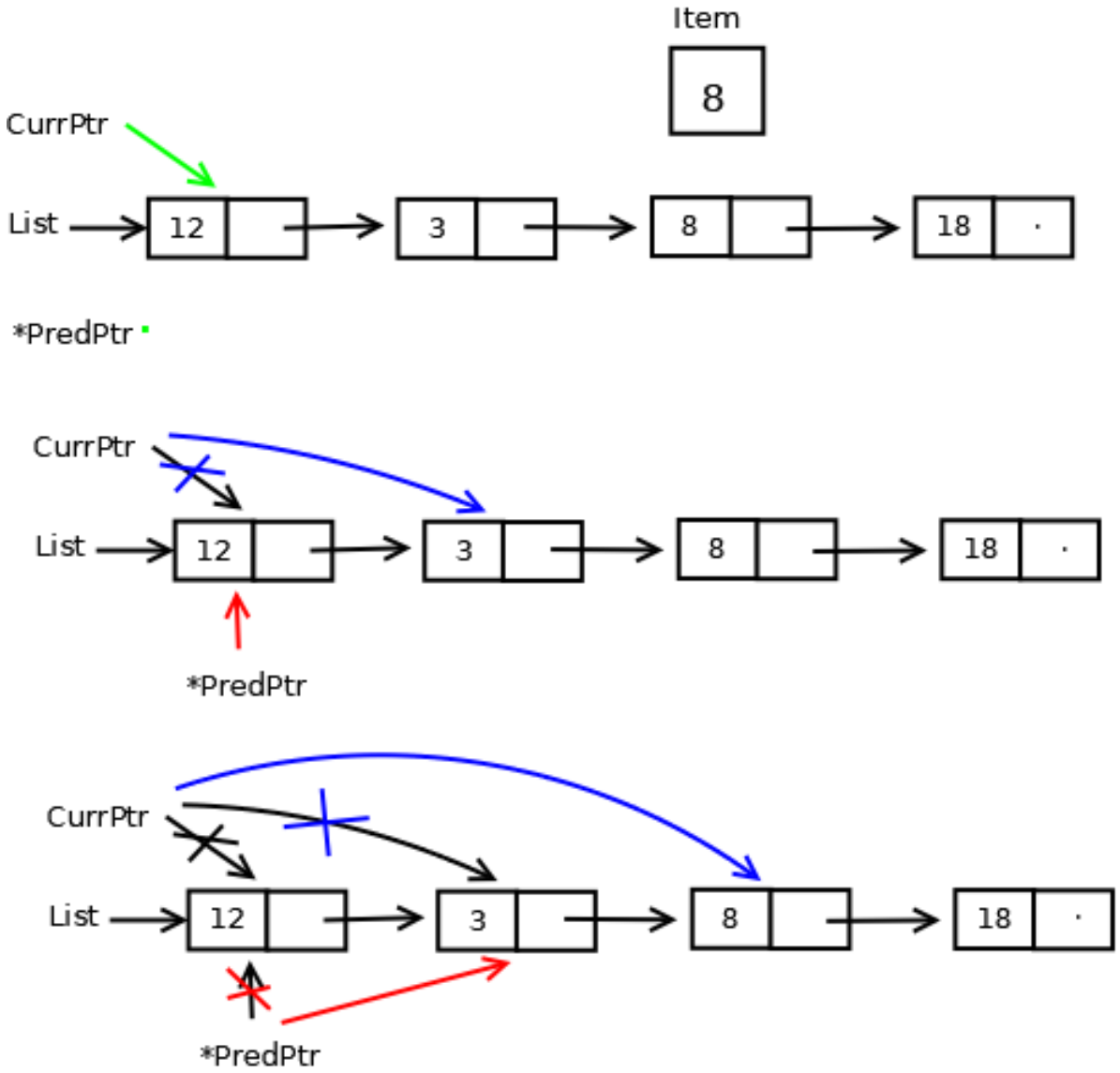
Σε αυτήν την περίπτωση πρόκειται για μη ταξινομημένη λίστα και το τρίτο βήμα στον **αλγόριθμο εισαγωγής** γίνεται απλά:

```
TempPtr = *List;
*List = TempPtr->Next;
```

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:

Αναζήτηση συνδεδεμένης λίστας (LinearSearch)




```
void LinearSearch (ListPointer List,
     ListElementType Item, ListPointer
     *PredPtr, boolean *Found)

  ListPointer CurrPtr;
  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;
```

Αναζήτηση συνδεδεμένης λίστας (LinearSearch)

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.*/

Αναζήτηση συνδεδεμένης λίστας (LinearSearch)

```
void LinearSearch (ListPointer List, ListElementType Item, ListPointer *PredPtr,
                   boolean *Found)
{
    ListPointer CurrPtr;
    boolean stop;                                  //χωρίς χρήση βοηθητικής stop

    CurrPtr = List;
    *PredPtr = NULL;
    stop = FALSE;                                  // *Found=FALSE;
    while (!stop && CurrPtr != NULL ) {            // while (!(*Found) && CurrPtr != NULL )
        if (CurrPtr->Data==Item )
                stop = TRUE;                       //*Found=TRUE;
        else
        {
                *PredPtr = CurrPtr;
                CurrPtr = CurrPtr->Next;
        }
    }
    *Found = stop;                                 // δε χρειάζεται αυτή η ανάθεση
}
```

Αλγόριθμος αναζήτησης ταξινομημένης σ. λίστας

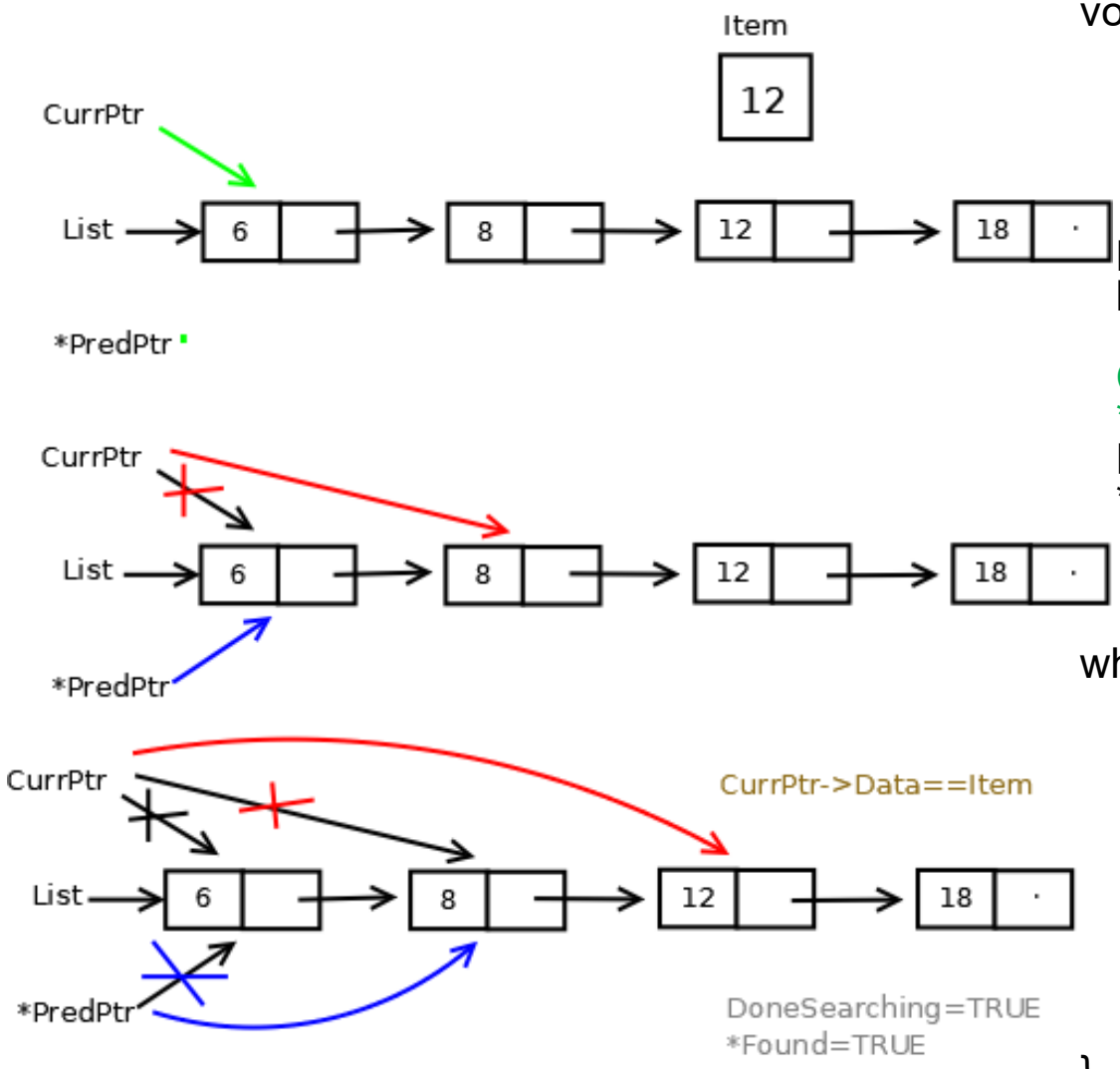
Σε μια ταξινομημένη λίστα οι κόμβοι είναι συνδεδεμένοι μεταξύ τους με τρόπο ώστε να επισκεπτόμαστε τα στοιχεία που είναι αποθηκευμένα στους κόμβους κατά **αύξουσα ή φθίνουσα σειρά** καθώς διασχίζουμε τη λίστα.

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.

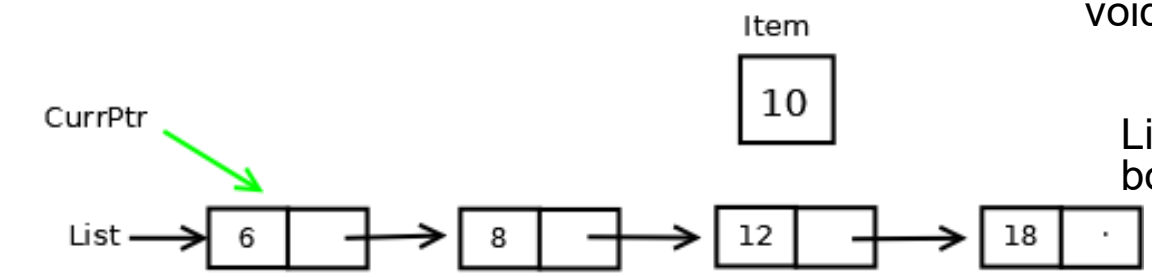
Ο αλγόριθμος αναζήτησης για μια ταξινομημένη συνδεδεμένη λίστα είναι ο ακόλουθος:

Αλγόριθμος αναζήτησης ταξινομημένης σ. λίστας

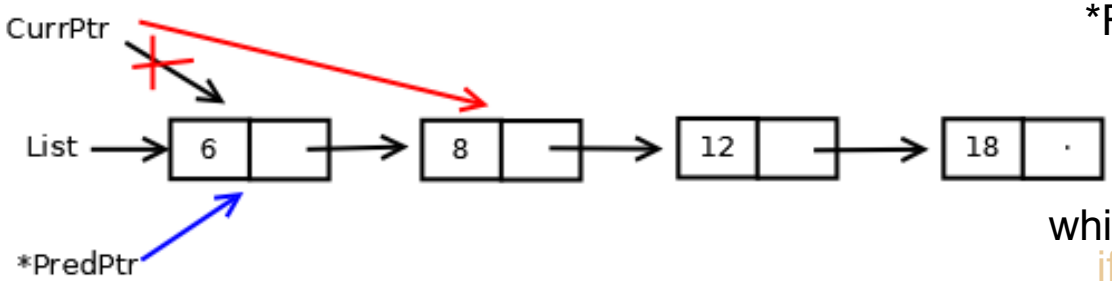


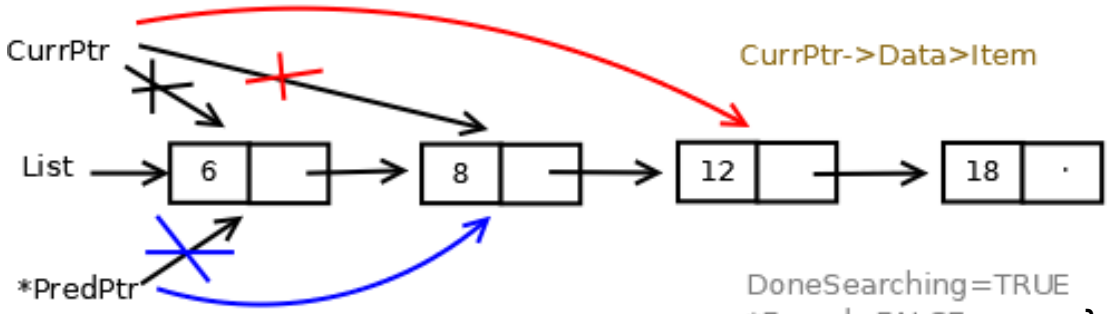

```
void OrderedLinearSearch (ListPointer List,
     ListElementType Item,
     ListPointer  *PredPtr, boolean *Found){

   ListPointer CurrPtr;
   boolean DoneSearching;

   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;
        }
    }
}
```

Αλγόριθμος αναζήτησης ταξινομημένης σ. λίστας




```
void OrderedLinearSearch (ListPointer List,
     ListElementType Item,
     ListPointer  *PredPtr, boolean *Found){
  ListPointer CurrPtr;
  boolean DoneSearching;

  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;
    }
  }
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

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.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

```
void OrderedLinearSearch(ListPointer List, ListElementType Item,
                         ListPointer  *PredPtr, boolean *Found)
{
    ListPointer CurrPtr;
    boolean DoneSearching;

    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;
      }
    }
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

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

typedef struct ListNode *ListPointer; /*ο τύπος των δεικτών για τους κόμβους*/
typedef struct ListNode
{
        ListElementType Data;
        ListPointer Next;
} ListNode;

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

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

```
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 OrderedLinearSearch(ListPointer List, ListElementType Item,
                  ListPointer *PredPtr, boolean *Found);
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

```
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;
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

```
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;

    CurrPtr = List;
    while (CurrPtr != NULL)
    {
        /*Εδώ παρεμβάλλονται οι απαραίτητες εντολές για την  επεξεργασία του
        CurrPtr->Data*/
        CurrPtr = CurrPtr->Next
    }
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

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.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

```
void LinearSearch (ListPointer List, ListElementType Item, ListPointer *PredPtr,
                   boolean *Found)
{
    ListPointer CurrPtr;
    boolean stop;                                  //χωρίς χρήση βοηθητικής stop

    CurrPtr = List;
    *PredPtr = NULL;
    stop = FALSE;                                  // *Found=FALSE;
    while (!stop && CurrPtr != NULL ) {            // while (!(*Found) && CurrPtr != NULL )
        if (CurrPtr->Data==Item )
                stop = TRUE;                       //*Found=TRUE;
        else
        {
                *PredPtr = CurrPtr;
                CurrPtr = CurrPtr->Next;
        }
    }
    *Found = stop;                                 // δε χρειάζεται αυτή η ανάθεση
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

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.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

```
void OrderedLinearSearch(ListPointer List, ListElementType Item,
                         ListPointer  *PredPtr, boolean *Found)
{
    ListPointer CurrPtr;
    boolean DoneSearching;

    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;
      }
    }
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

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*.*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

```
void LinkedInsert(ListPointer *List, ListElementType Item,
                  ListPointer PredPtr)
{
    ListPointer TempPtr;

   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;
    }
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

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*/

Πακέτο για τον ΑΤΔ Συνδεδεμένη Λίστα με δείκτες

```
void LinkedDelete(ListPointer *List, ListPointer PredPtr)
{
    ListPointer TempPtr;
    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);
    }
}
```

Κλήση από το πρόγραμμα πελάτης …main()

```
typedef int ListElementType;          /* ο τύπος των στοιχείων της συνδεδεμένης λίστας
                                         ενδεικτικά τύπου int */
typedef struct ListNode *ListPointer;  //ο τύπος των δεικτών για τους κόμβους
typedef struct ListNode
{
     ListElementType Data;
    ListPointer Next;
} ListNode;

typedef enum {
    FALSE, TRUE
} boolean;
void CreateList(ListPointer *List);
…
main()
{
   ListPointer AList;

   CreateList(&AList);
}
```

Κλήση από το πρόγραμμα πελάτης …main()

```
…
void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
…
main()
{
   ListPointer AList;

    CreateList(&AList);
    if (EmptyList(AList)) printf("Empty List\n");
}
```

Κλήση από το πρόγραμμα πελάτης …main()

```
…
void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
void LinkedTraverse(ListPointer List);
…
main()
{
   ListPointer AList;

    CreateList(&AList);
    if (EmptyList(AList)) printf("Empty List\n");
    LinkedTraverse(AList);
}
```

Κλήση από το πρόγραμμα πελάτης …main()

```
…
void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
void LinkedTraverse(ListPointer List);
void LinearSearch(ListPointer List, ListElementType Item,
          ListPointer *PredPtr, boolean *Found);
…
main()
{
    ListPointer AList;
    ListElementType keyvalue;
    boolean Found;

    CreateList(&AList);
    if (EmptyList(AList)) printf(“Empty List\n”);
    LinkedTraverse(AList);
    printf("Enter Data for searching: "); scanf("%d", &keyvalue);
    LinearSearch(AList, keyvalue, &PredPtr, &Found);
    if ( Found ) printf("Found IN NODE \n");
    else printf("Item NOT IN LIST\n");
```

Κλήση από το πρόγραμμα πελάτης …main()

```
…
void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
void LinkedTraverse(ListPointer List);
void LinearSearch(ListPointer List, ListElementType Item,
            ListPointer *PredPtr, boolean *Found);
void OrderedLinearSearch(ListPointer List, ListElementType Item,
    ListPointer *PredPtr, boolean *Found);
…
main()
{
    ListPointer Alist, PredPtr;
    ListElementType keyvalue;
    boolean Found;
    ….
     printf("Enter Data for searching: "); scanf("%d", &keyvalue);
    OrderedLinearSearch(AList, keyvalue, &PredPtr, &Found);
    if ( Found ) printf("Found IN NODE \n");
    else printf("Item NOT IN LIST\n");
```

Κλήση από το πρόγραμμα πελάτης …main()

```
…
void LinkedInsert(ListPointer *List, ListElementType Item,
    ListPointer PredPtr);
void OrderedLinearSearch(ListPointer List, ListElementType Item,
    ListPointer *PredPtr, boolean *Found);
…
main()
{
    ListPointer Alist, PredPtr;
    ListElementType keyvalue;
    boolean Found;
    int choice1;
    ….
     printf("Enter a number for insertion : "); scanf("%d", &keyvalue);
       if (choice1==1)
           OrderedLinearSearch(AList, keyvalue, &PredPtr, &Found);
       else
          PredPtr= NULL;
      LinkedInsert(&AList, keyvalue, PredPtr);
}
```

Κλήση από το πρόγραμμα πελάτης …main()

```
…
void LinkedDelete(ListPointer *List, ListPointer PredPtr);
void LinearSearch(ListPointer List, ListElementType Item, ListPointer
    *PredPtr, boolean *Found);
…
main()
{
    ListPointer Alist, PredPtr;
    ListElementType keyvalue;
    boolean Found;
    ….
     printf("Enter Data for deleting: "); scanf("%d", &keyvalue);
     LinearSearch(AList, keyvalue, &PredPtr, &Found);
     if (Found)
          LinkedDelete(&AList, PredPtr);
    else
          printf("Item NOT IN LIST\n");
}
```