

# **ΛΙΣΤΕΣ**

## **4.1 Ο ΑΤΔ Λίστα & Υλοποίηση Λίστας με σειριακή αποθήκευση**

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

## 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.

## 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.

### 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

## 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.

## Εισαγωγή στοιχείου (Insert)

**Δέχεται:** Μια λίστα, ένα στοιχείο δεδομένων και μια θέση μέσα στη λίστα.

**Λειτουργία:** Εισάγει το στοιχείο στην συγκεκριμένη θέση της λίστας.

**Επιστρέφει:** Την τροποποιημένη λίστα.

## Διαγραφή στοιχείου (Delete)

**Δέχεται:** Μια λίστα και μια θέση μέσα στη λίστα.

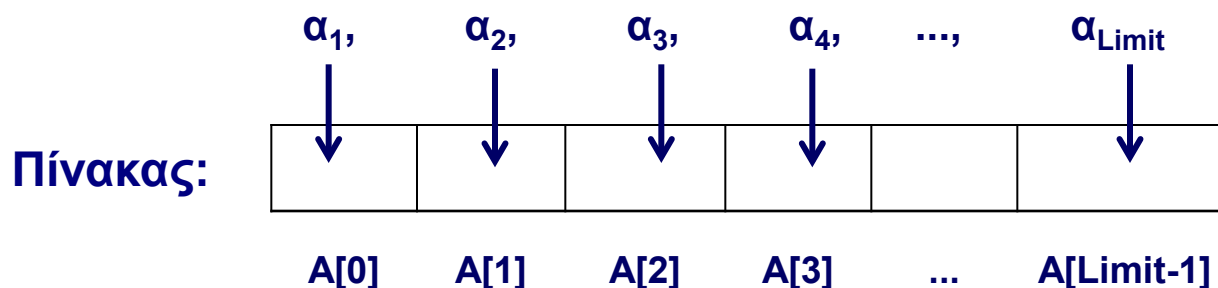
**Λειτουργία:** Διαγράφει το στοιχείο από την συγκεκριμένη θέση της λίστας.

**Επιστρέφει:** Την τροποποιημένη λίστα.

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 0 του πίνακα, το δεύτερο στην θέση 1, κ.ο.κ.

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



# Δομές δεδομένων

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

```
#define ListLimit 50
```

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

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

```
} ListType;
```

```
....
```

```
ListType List; //δήλωση μεταβλητής List για τη διαχείριση λίστας
```

List	Size	πίνακας Element[ ] κάθε στοιχείο του τύπου ListElementType							
		0	1	2					ListLimit-1
	2	4	12	?	?	?	?	?	...

# Δομές δεδομένων

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 (**List.Size==0**) είναι αληθής ή ψευδής αντίστοιχα.
- 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



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	...

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:

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	...

List

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

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.

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.

# Αλγόριθμος εισαγωγής στοιχείου σε λίστα (insert)

Αν γράψουμε ένα αλγόριθμο για τη λειτουργία της εισαγωγής, τότε θα πρέπει πρώτα να ελέγχει αν ο πίνακας είναι γεμάτος και, αν δεν είναι, να εκτελεί τις απαιτούμενες μετατοπίσεις των στοιχείων του πίνακα πριν εισάγει το νέο στοιχείο σ' αυτόν.

Στην περίπτωση που ο πίνακας είναι γεμάτος, είναι προφανές ότι το νέο στοιχείο δεν μπορεί να εισαχθεί στον πίνακα.

Στη συνέχεια παρουσιάζεται η υλοποίηση του αλγόριθμου για την εισαγωγή στοιχείου σε λίστα (Insert):

# Διαδικασία εισαγωγής στοιχείου σε λίστα (Insert)

**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*=-1).

Επιστρέφει: Την τροποποιημένη λίστα.

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

{

**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++;

}

}

Μετατόπισε τα στοιχεία του πίνακα *Element* από τη θέση *i* στη θέση *i+1*, δηλαδή κατά μία θέση δεξιότερα, προς το τέλος του πίνακα

Θέσε στη θέση *Pos + 1* του πίνακα *Element* την τιμή της *Item*

# Παράδειγμα εισαγωγής στοιχείου σε λίστα (Insert)

		Pos					Size-1	
		↓					↓	
			List					
θέση	0	1	2	3	4	5	6	...
αριθμός	4	12	37	40	49	63	71	...

			Pos+1	→				Size-1	
			↓	List				↓	
θέση	0	1	2	3	4	5	6	7	...
αριθμός	4	12	28	37	40	49	63	71	...

Item

28

Pos

1

Size

7

Η θέση Pos θα πρέπει να εντοπιστεί πριν την κλήση της Insert

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

```
{  
    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++;  
    }  
}
```

# Αποδοτικότητα της διαδικασίας Insert

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.

- 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.

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.

## Παράδειγμα (... συνέχεια)

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	...

Και η λειτουργία της διαγραφής ενός στοιχείου από τη λίστα, δηλαδή, μπορεί να προκαλέσει μετατόπιση κάποιων στοιχείων της.

Στη συνέχεια παρουσιάζεται ένας **αλγόριθμος για τη διαγραφή ενός στοιχείου από μια λίστα (Delete)**:



# Διαδικασία διαγραφής στοιχείου από λίστα (delete)

**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.\**

```
{  
    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--;  
    }  
}
```

*Μετατόπισε τα στοιχεία του πίνακα *Element* από τη θέση  $i + 1$  στη θέση  $i$ , δηλαδή κατά μια θέση αριστερότερα προς την αρχή του πίνακα*

# Παράδειγμα διαγραφής στοιχείου από λίστα (delete)

		Pos ↓						Size-1 ↓	
			List						
θέση	0	1	2	3	4	5	6	7	...
αριθμός	4	12	28	37	40	49	63	71	...

Size

8

Pos

1

Η θέση Pos θα πρέπει να εντοπιστεί πριν την κλήση της Delete



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

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

```
{  
    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--;  
    }  
}
```

# Διαδικασία διαγραφής στοιχείου από λίστα (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  $n/2$ .

# Πακέτο για τον ΑΤΔ Λίστα με πίνακα

*// Filename ListADT.h*

```
#define ListLimit 50
```

```
typedef int ListElementType;
```

```
typedef struct {
```

```
    int Size;
```

```
    ListElementType Element[ListLimit];
```

```
} ListType;
```

```
typedef enum {
```

```
    FALSE, TRUE
```

```
} boolean;
```

```
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);
```

# Πακέτο για τον ΑΤΔ Λίστα με πίνακα

// Filename *ListADT.c*

**void** CreateList(ListType \*List)

/\*Λειτουργία:       Δημιουργεί μια κενή λίστα.

Επιστρέφει:       Μια κενή λίστα\*/

{

    (\*List).Size = 0;

}

**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);

}

```
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));
```

```
}
```

# Πακέτο για τον ΑΤΔ Λίστα με πίνακα

**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*=-1).

Επιστρέφει: Την τροποποιημένη λίστα.

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

```
{  
    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++;  
    }  
}
```

# Πακέτο για τον ΑΤΔ Λίστα με πίνακα

**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.\*/\**

```
{  
    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--;  
    }  
}
```



# Πακέτο για τον ΑΤΔ Λίστα με πίνακα

```
void TraverseList(ListType List)
```

```
/*Δέχεται:      Μια λίστα List.
```

```
Λειτουργία:      Κάνει διάσχιση της λίστα List, αν δεν είναι κενή.
```

```
Έξοδος:          Εξαρτάται από την επεξεργασία.*/  
{
```

```
    int i;
```

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

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

```
    else
```

```
    {
```

```
        printf("\nPlithos stoixeiwn sth lista %d\n",List.Size);
```

```
        for (i=0; i<List.Size; i++)
```

```
            printf("%d\n",List.Element[i]);
```

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

```
    }
```

```
}
```

Επειδή, λοιπόν, κάθε φορά που εισάγουμε ένα στοιχείο στη λίστα ή διαγράφουμε ένα από αυτήν, είναι πιθανό να προκαλέσουμε μετατόπιση πολλών στοιχείων μέσα σ' αυτήν, οι διαδικασίες αυτές μπορεί να είναι πολύ αργές.

Η υλοποίηση που περιγράφηκε παραπάνω είναι κατάλληλη για **στατικές λίστες** (static lists) όχι όμως και για **δυναμικές λίστες** (dynamic lists), στις οποίες εκτελείται ένας μεγάλος αριθμός εισαγωγών και διαγραφών.

Όταν οι εισαγωγές και διαγραφές χρειάζεται να γίνονται σε οποιαδήποτε θέση της λίστας, τότε καταφεύγουμε στις **συνδεδεμένες λίστες** (linked lists).