Προγραμματισμός Υπολογιστών

Λίστα Δεδομένων

Νικόλαος Ζ. Ζάχαρης
Καθηγητής

Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Συνδεδεμένη Λίστα (Linked List)

Οι πίνακες αποτελούν ένα εύκολο και γρήγορο τρόπο αποθήκευσης δεδομένων. Το πρόβλημα με τους πίνακες είναι ότι έχουν προκαθορισμένο μήκος με αποτέλεσμα :

Α) να έχουμε δεσμευμένες θέσεις μνήμης την στιγμή που δεν περιέχουν χρήσιμα δεδομένα

Π.χ. Μια εφαρμογή έχει ένα πίνακα με 100 θέσεις μνήμης για την αποθήκευση των στοιχείων των πελατών. Αν έχουμε κάνει εισαγωγή 10 πελάτες τότε διατηρούμε 90 θέσεις στον πίνακα χωρίς χρήσιμα δεδομένα.

Β) όταν γεμίσουμε όλες τις θέσεις να απαιτείται ένας νέος πίνακας (ή αρχείο) για την μεταφορά των δεδομένων.

Π.χ. Μια εφαρμογή έχει ένα πίνακα με 100 θέσεις μνήμης για την αποθήκευση των στοιχείων των πελατών. Αν έχουμε γεμίσει όλες τις θέσεις τότε θα πρέπει να αποθηκεύσουμε τα υπάρχοντα στοιχεία σε ένα αρχείο ή άλλο πίνακα, να ελευθερώσουμε την μνήμη του πίνακα και να δεσμεύσουμε νέα μεγαλύτερη χωρητικότητα και εν συνεχεία να ξαναγεμίσουμε το πίνακα με τα στοιχεία από το αρχείο. Είτε χρησιμοποιήσουμε άλλο πίνακα ή αρχείο έχουμε καθυστέρηση στην λειτουργία του προγράμματος.

Η λύση είναι η χρήση μιας συνδεδεμένης λίστας. Πρακτικά η λειτουργία έχει ως εξής :

Με την έναρξη της εφαρμογής δεν έχουμε δεσμεύσει καθόλου μνήμη. Όταν κάνουμε εισαγωγή ένα νέο πελάτη τότε δεσμεύουμε μνήμη ακριβώς για τα στοιχεία ενός πελάτη. Έστω η μνήμη που δεσμεύσαμε αρχίζει από την θέση Α. Όταν κάνουμε εισαγωγή ένα δεύτερο πελάτη τότε δεσμεύουμε μνήμη ακριβώς για τα στοιχεία ενός πελάτη. Έστω η μνήμη που δεσμεύσαμε αρχίζει από την θέση Β.

Στην ουσία αυτές οι δύο περιοχές της μνήμης είναι το πελατολόγιο μας οπότε για να συνδέσουμε τα στοιχεία μεταξύ τους θα πρέπει μέσα στην κάθε δομή που περιέχει τα στοιχεία ενός πελάτη να υπάρχει και ένας δείκτης που να δείχνει στην επόμενη δεσμευμένη θέση μνήμης. Κατά αυτό το τρόπο δημιουργούμε μια αλυσίδα από θέσεις μνήμης.

```
typedef struct Pelatis {
    char name [30];
    char phone[20];
    struct Pelatis *next;
} Pelatis;
```

Τα στοιχεία ενός πελάτη

Ένας δείκτης στην επόμενη θέση του πελατολογίου

Το σύνολο των δεδομένων που πρέπει να διατηρούμε για την κάθε θέση του πελατολογίου.

Οπότε το πελατολόγιο της εφαρμογής είναι ένας δείκτης, ο οποίος αρχικά είναι NULL.

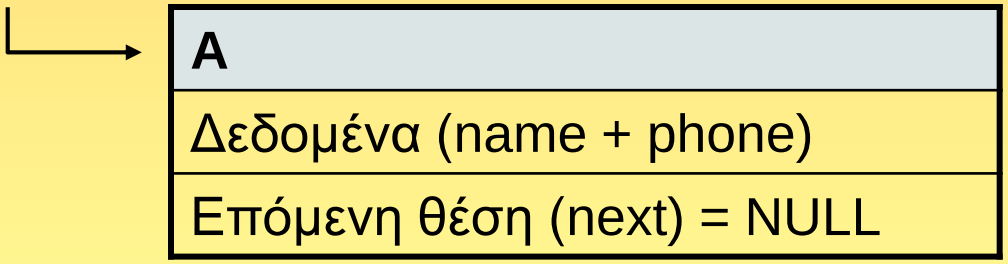
Pelatis *Pelatologio = NULL;

Όταν κάνουμε εισαγωγή του πρώτου πελάτη τότε ο δείκτης Pelatologio δείχνει στην θέση Α και ο δείκτης μέσα στην δομή για την επόμενη θέση του πελατολογίου είναι NULL.

Pelatologio = NULL

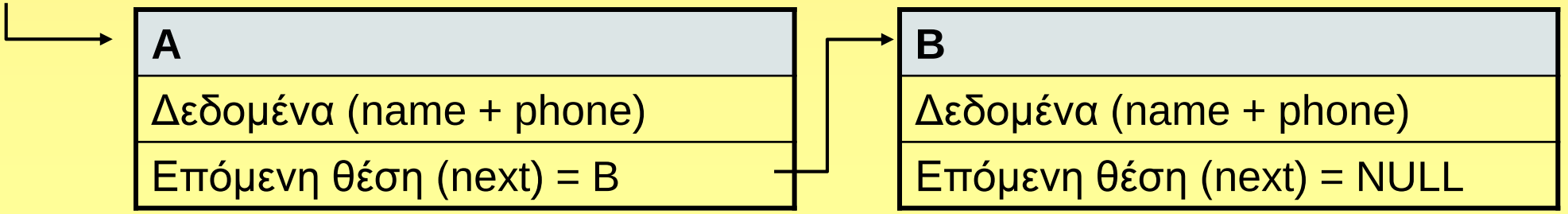
Αρχικά

Pelatologio



Εισαγωγή πρώτου πελάτη

Pelatologio



Εισαγωγή δεύτερου πελάτη

Η συνδεδεμένη λίστα προσομοιάζει με μια αλυσίδα όπου ο κάθε κρίκος είναι και ένα στοιχείο. Επίσης κάθε στοιχείο συνδέεται με το επόμενο του και έτσι δημιουργείται μια σειριακή σχέση αφού για να φτάσουμε στο δεύτερο στοιχείο θα πρέπει να διασχίσουμε το πρώτο στοιχείο. Ο δείκτης Pelatologio δείχνει πάντα την θέση μνήμης του πρώτου στοιχείου με σκοπό την διάσχιση όλων των στοιχείων της λίστας.

Στην συνδεδεμένη λίστα, κάθε νέο στοιχείο προστίθεται στο τέλος αυτής. Έτσι λοιπόν για κάθε νέο στοιχείο θα πρέπει να αρχίσουμε από την αρχή της λίστας και να την διασχίσουμε έως του σημείου που ο δείκτης για την επόμενη θέση ενός κόμβου να είναι NULL. Αυτή είναι και η τελευταία θέση της λίστας. Αυτός ο κόμβος από τελευταίος στην λίστα θα γίνει προτελευταίος και ο δείκτης του για την θέση μνήμης του επόμενου στοιχείου από τιμή NULL θα αλλάξει με τη θέση μνήμης του νέου στοιχείου, το οποίο με την σειρά του θα γίνει το τελευταίο στοιχείο στην λίστα.

Η διάσχιση όλων των στοιχείων της λίστας με σκοπό την εύρεση της τελευταίας θέσης κάθε φορά που θέλουμε να προσθέσουμε ένα νέο στοιχείο είναι μια αρκετά χρονοβόρα διαδικασία η οποία απλοποιείται με την διατήρηση, καθ' όλη την διάρκεια της εφαρμογής, ενός δείκτη στο τελευταίο στοιχείο της λίστας.

Από τα προηγούμενα γίνεται κατανοητό ότι θα πρέπει να διατηρούμε δύο δείκτες ένα στην αρχή της λίστας και έναν στην τελευταία θέση. Η πρώτη θέση είναι η αρχή του πελατολογίου και θα την χρησιμοποιούμε για την διάσχιση όλων των στοιχειών ενώ ο δείκτης στην τελευταία θέση θα χρησιμοποιείται για την γρήγορη εισαγωγή νέων δεδομένων.

```
typedef struct Pelatis {
    char name [30];
    char phone[20];
    struct Pelatis *next;
} Pelatis;

Pelatis *FirstPelatis = NULL;
Pelatis *LastPelatis = NULL;

void showList(void) {
   Pelatis* temp = FirstPelatis;

   while( temp != NULL) {
      printf("%s\n", temp->name);
      temp = temp->next;
  }
}
```

Η κάθε δομή μεταξύ των άλλων στοιχείων περιέχει και ένα δείκτη για την επόμενη θέση μνήμης.

Για την διαχείριση της λίστας χρειαζόμαστε δύο δείκτες, έναν στην πρώτη θέση και ένα στην τελευταία θέση της λίστας. Αρχικά και οι δύο δείκτες είναι NULL.

Για την διάσχιση χρειαζόμαστε ένα δείκτη (temp) που δείχνει στην πρώτη θέση της λίστας. Εφόσον ο δείκτης είναι διάφορος του NULL τότε επιδεικνύουμε τα στοιχεία της τρέχουσας θέσης και ο δείκτης συνεχίζει στην επόμενη θέση της λίστας.

Εισαγωγή ενός πελάτη στη λίστα

```
void AddNewPelatis(void) {
    char name[20];
    Pelatis* temp;
    temp = (Pelatis *) malloc( sizeof(Pelatis));
  if(temp == NULL) {
     printf("\nError : No enough memory.\n");
     return;
  }

    printf("\nType pelatis name : ");
    scanf("%s", name);

    strcpy(temp->name, name);
    temp->next = NULL;

    if(FirstPelatis == NULL)  {
        FirstPelatis = temp;
        LastPelatis = temp;
    }
    else {
        LastPelatis->next = temp;
        LastPelatis = temp;
    }
}
```

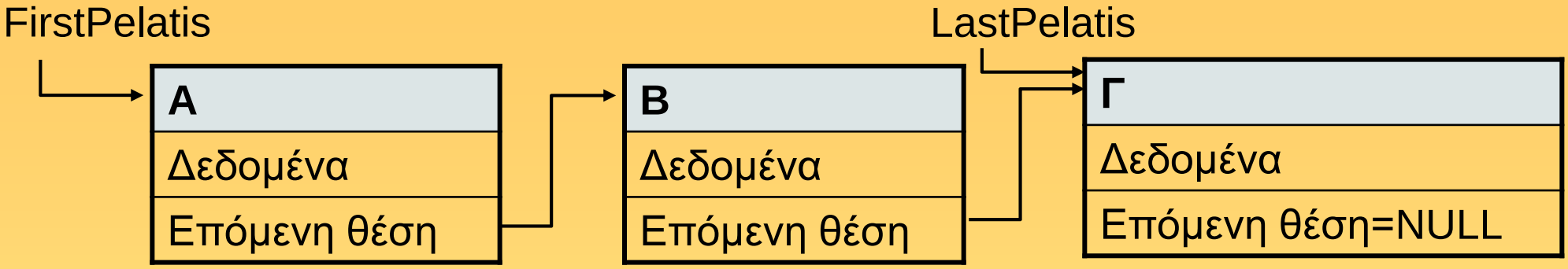
Για την εισαγωγή ενός νέου πελάτη χρειαζόμαστε ένα δείκτη (temp) για τον οποίο δεσμεύουμε μνήμη ικανή να αποθηκεύσουμε τα στοιχεία ενός πελάτη. Αν δεν υπάρχει διαθέσιμη μνήμη τότε το σύστημα επιστρέφει την τιμή NULL.

Αποθηκεύουμε τα στοιχεία του πελάτη στην μνήμη που μόλις δεσμεύσαμε. Ο δείκτης για την επόμενη θέση είναι NULL αφού μετά την προσθήκη θα γίνει η τελευταία θέση του πελατολογίου.

Αν το πελατολόγιο είναι άδειο τότε και οι δύο δείκτες ΑΡΧΙΚΗ και ΤΕΛΙΚΗ θέση θα δείχνουν στην ίδια θέση μνήμης αφού θα υπάρχει μόνο ένα στοιχείο..

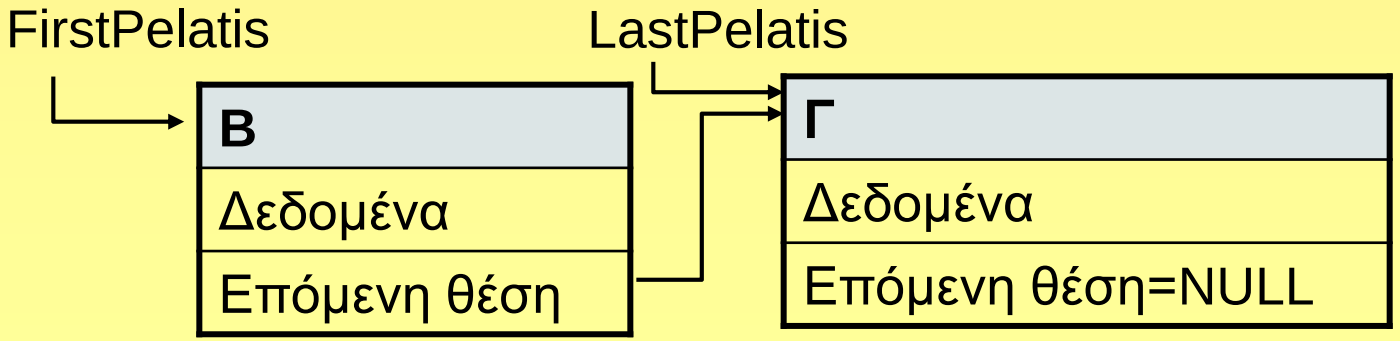
Αν το πελατολόγιο δεν είναι άδειο τότε ο δείκτης για την επόμενη θέση του τελευταίου στοιχείου θα δείχνει στην μνήμη που μόλις δεσμεύσαμε, η οποία θα γίνει η τρέχουσα τελευταία θέση.

Διαγραφή ενός πελάτη από τη λίστα

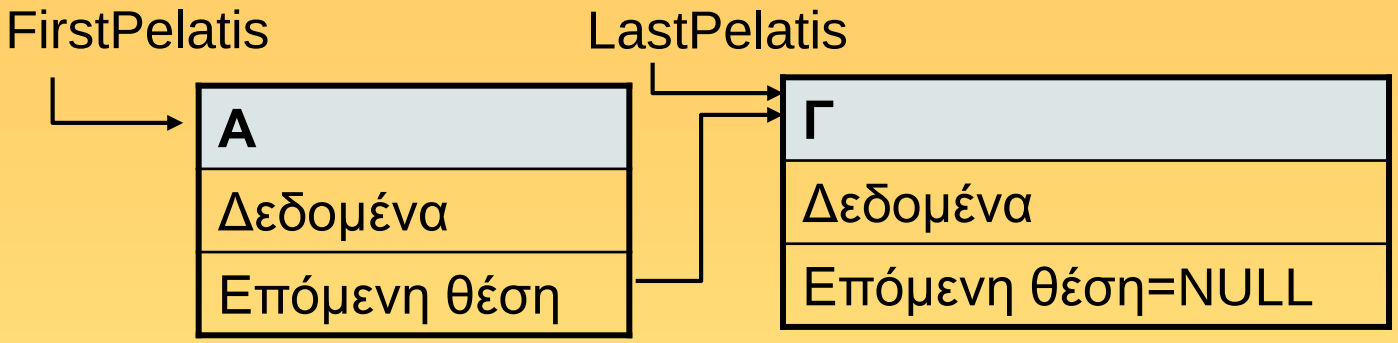


Στην διαγραφή ενός στοιχείου από την λίστα θα πρέπει να κάνουμε αναδιάταξη των εσωτερικών δεικτών. Επίσης θα πρέπει να χρησιμοποιήσουμε ένα δείκτη που θα δείχνει στο στοιχείο που θα διαγράψουμε ώστε μετά την αναδιάταξη να ελευθερώσουμε την μνήμη. Υπάρχουν τρεις περιπτώσεις :

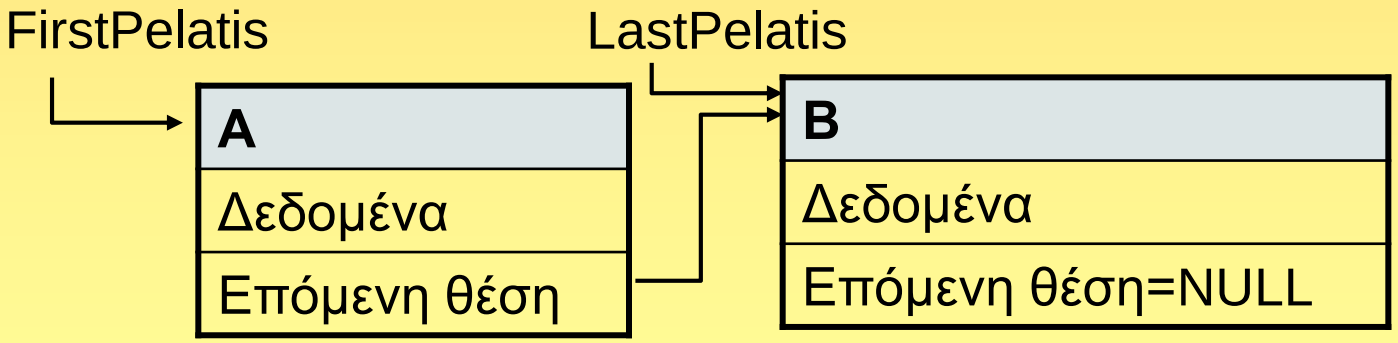
Διαγραφή του Α – Σε αυτή την περίπτωση ο FirstPelatis θα δείχνει στην θέση μνήμης Β.



Διαγραφή του Β – Σε αυτή την περίπτωση ο δείκτης του Α θα δείχνει στην θέση μνήμης Γ.



Διαγραφή του Γ – Σε αυτή την περίπτωση ο δείκτης του Β για την επόμενη θέση μνήμης θα πάρει την τιμή NULL και ο δείκτης LastAccount θα δείχνει στην θέση μνήμης Β.



Η περίπτωση Γ είναι υποπερίπτωση της Β αφού κάνουμε τις ίδιες ενέργειες με επιπλέον ενέργεια την επανατοποθέτηση του δείκτη LastPelatis στο προηγούμενο στοιχείο της λίστας.

Τα στοιχεία στην λίστα είναι σειριακά τοποθετημένα οπότε κατασκευάζουμε μια συνάρτηση FindPelatis η οποία δέχεται μια θέση στην λίστα και επιστρέφει την αντίστοιχη διεύθυνση. Για το προηγούμενο παράδειγμα, η FindPelatis με παράμετρο 1 επιστρέφει Α, με παράμετρο 2 επιστρέφει Β και με 3 επιστρέφει Γ. Για οποιαδήποτε άλλη τιμή επιστρέφει NULL.

```
Pelatis *FindPelatis(int pos){
   int index = 0;
   Pelatis* temp = FirstPelatis;

   while( temp != NULL) {
     index = index + 1;
            if(index == pos)
               return temp;
            else
               temp = temp->next;
   }

    return NULL;
}
```

Χρησιμοποιούμε ένα δείκτη ο οποίος αρχικοποιείται με τιμή την πρώτη θέση στην λίστα.

Εφόσον ο δείκτης είναι διάφορος του NULL τότε αυξάνουμε ένα μετρητή και τον συγκρίνουμε με την θέση που αναζητούμε στην λίστα. Αν οι δύο τιμές είναι ίσες τότε επιστρέφουμε την τρέχουσα θέση μνήμης διαφορετικά ο δείκτης προχωρά στην επόμενη θέση της λίστας.

```
void DeletePelatis(int pos)  {
   if(pos == 1)  {
         if(FirstPelatis != NULL) {
            Pelatis *temp = FirstPelatis;
            FirstPelatis = FirstPelatis->next;
            free(temp);
            temp = NULL;
         }
        return;
      }
    if(pos > 1) {
         Pelatis* current  = FindPelatis (pos);
         Pelatis* previous = FindPelatis (pos-1);
         if((current != NULL) && (previous != NULL))  {
            previous->next = current->next;

             if(current-> next == NULL)
               LastPelatis = previous;
            free(current);
            current = NULL;
         }
         else {
             printf("\nError : Not a valid number.\n");
         }
      }
}
```

Διαγραφή του Α – Σε αυτή την περίπτωση ο FirstPelatis θα δείχνει στην επόμενη θέση μνήμης.

Διαγραφή του Β ή Γ – Εύρεση των δεικτών της θέσης προς διαγραφή καθώς και της προηγού- μενης της.

Αναδιάταξη των δεικτών

Διαγραφή του Γ – Αναδιάταξη της τελευταίας θέσης της λίστας.