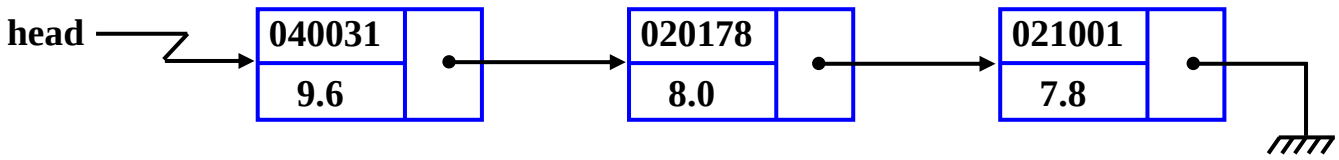
ΣΥΝΔΕΔΕΜΕΝΕΣ ΛΙΣΤΕΣ (Linked Lists)

Η συνδεδεμένη λίστα είναι μία διατεταγμένη λίστα από κόμβους. Ο κάθε κόμβος είναι μια δομή που περιέχει πεδία δεδομένων (*π.χ. data1, data2, ..., datan*) και μία σύνδεση (*link*), δηλαδή ένα πεδίο στο οποίο αποθηκεύεται η διεύθυνση του επόμενου κόμβου. Ο πρώτος κόμβος δεικτοδοτείται από έναν ειδικό pointer (κεφαλή της λίστας ή head) ενώ ο τελευταίος κόμβος περιέχει μία "γειωμένη" σύνδεση (δηλαδή, τη μηδενική διεύθυνση) που συμβολίζεται με το NULL (στο <stdio.h> υπάρχει ο ορισμός: `#define NULL 0`) και σηματοδοτεί το τέλος της λίστας.

Γιατί να μην χρησιμοποιούμε πίνακες αντί για συνδεδεμένες λίστες αφού και ένας πίνακας είναι, επίσης, κατάλληλος για την αποθήκευση μιας διατεταγμένης λίστας δεδομένων; Το μειονέκτημα του πίνακα (απλού ή σύνθετου πίνακα δομών) είναι ότι τα στοιχεία του καταλαμβάνουν συνεχόμενες θέσεις μνήμης και το μέγεθος της διατεταγμένης λίστας θα είναι καθορισμένο από τη διάσταση του πίνακα. Κατά συνέπεια, η χρήση πινάκων δεν ενδείκνυται σε εφαρμογές που το πλήθος των δεδομένων είναι μεταβαλλόμενο ενώ είναι και αναποτελεσματική στην περίπτωση εισαγωγής ή διαγραφής δεδομένων.

Τα απαραίτητα στοιχεία για τη δημιουργία συνδεδεμένων λιστών είναι τα εξής: α) μηχανισμός για τον ορισμό της δομής ενός κόμβου (χρήση της `struct`), β) τρόπος δημιουργίας νέων κόμβων σύμφωνα με τις ανάγκες (χρήση της `malloc()`), και γ) δυνατότητα καταστροφής κόμβων που δεν χρησιμοποιούνται (χρήση της `free()`).

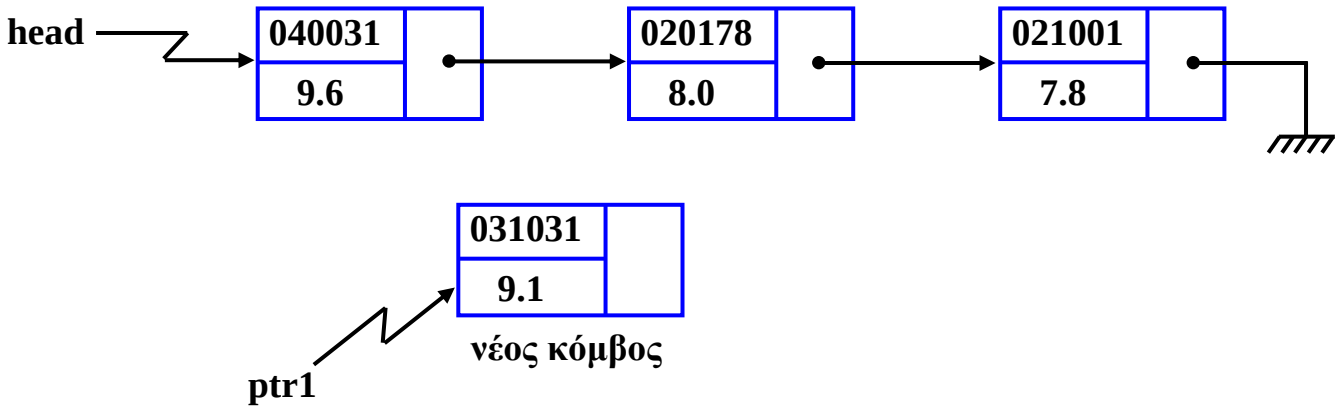
<u>Παράδειγμα λίστας</u> (πεδία κόμβων: Αρ. Μητρώου, βαθμός, link)



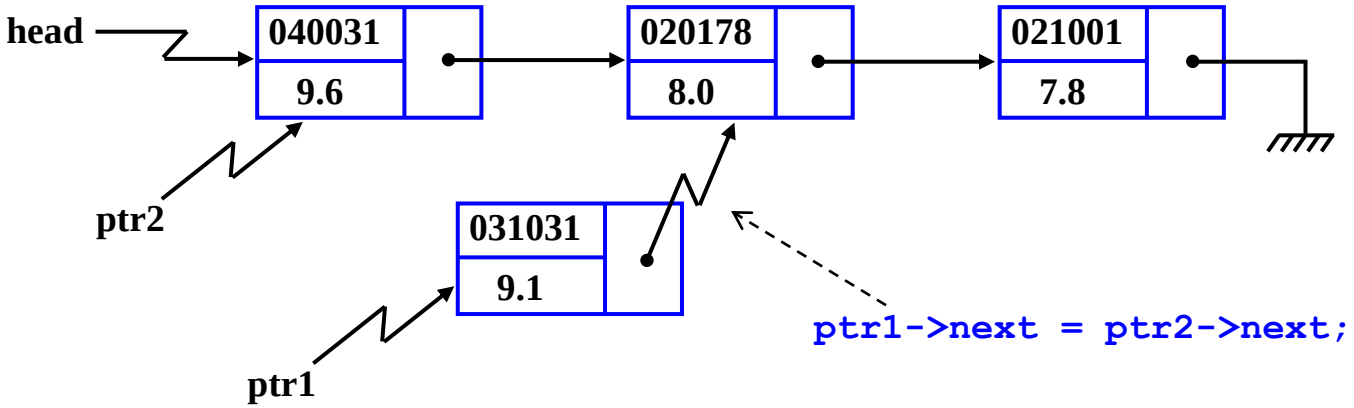
```
#include <stdio.h>

struct node {
     char AM[7];                /* Αριθμός Μητρώου */
     float grade;               /* Βαθμός */
     struct node *next;         /* Pointer στον επόμενο κόμβο */
};

main()
{
  struct node *head;
  struct node *CreateList();

  head = CreateList();   /* Δημιουργία αρχικής λίστας */
  ...
}

#include <ctype.h>
#include <string.h>
#define STOP 'N'

struct node *CreateList()
{
  int count = 0;
  float bathmos;
  char  ch, am[7], junk[40];
  struct node *h, *p;

  printf("Do you wish to insert a new node? (Y/N)");
  scanf("%c",&ch);
  gets(junk);        /* Άδειασε το buffer */
  ch = toupper(ch); /* Αν ch == 'n' τότε μετατρέπουμε σε 'N' */
```

```
  while (ch != STOP)
  {
    printf("AM (6 digits): \n");
    scanf("%s",am);
    printf("Bathmos: \n");
    scanf("%f",&bathmos);
    if (count == 0)    /* Αρχικοποιούμε τον head-pointer h */
    {
      h = (struct node *) malloc(sizeof(struct node));
          /* Δεσμεύει χώρο για τη δομή node και επιστρέφει
             pointer στη δομή */
      strcpy(h->AM,am);  /* Αντιγράφει το string am στο πεδίο AM
                            του κόμβου στον οποίο δείχνει ο h */
      h->grade = bathmos;
      p = h;     /* Θέτουμε και τον pointer p να δείχνει
                    στον κόμβο που δείχνει ο h */
    }
    else    /* Δεν πειράζουμε τον h. Χρησιμοποιούμε τον p */
    {
      p->next = (struct node *) malloc(sizeof(struct node));
         /* Συνδέει τον προηγούμενο κόμβο στον οποίο δείχνει ο p
            με τον τωρινό μέσω του πεδίου next */
      p = p->next;    /* Ο p τώρα δείχνει στον τωρινό κόμβο */
      strcpy(p->AM,am);
      p->grade = bathmos;
    }
    printf("Do you wish to insert a new node? (Y/N)");
    scanf("%c",&ch);
    gets(junk);
    ch = toupper(ch);
    count++;
  }
  p->next = NULL;     /* Τερματισμός λίστας */
  return(h);          /* Επιστρέφουμε τον head-pointer */
}
```

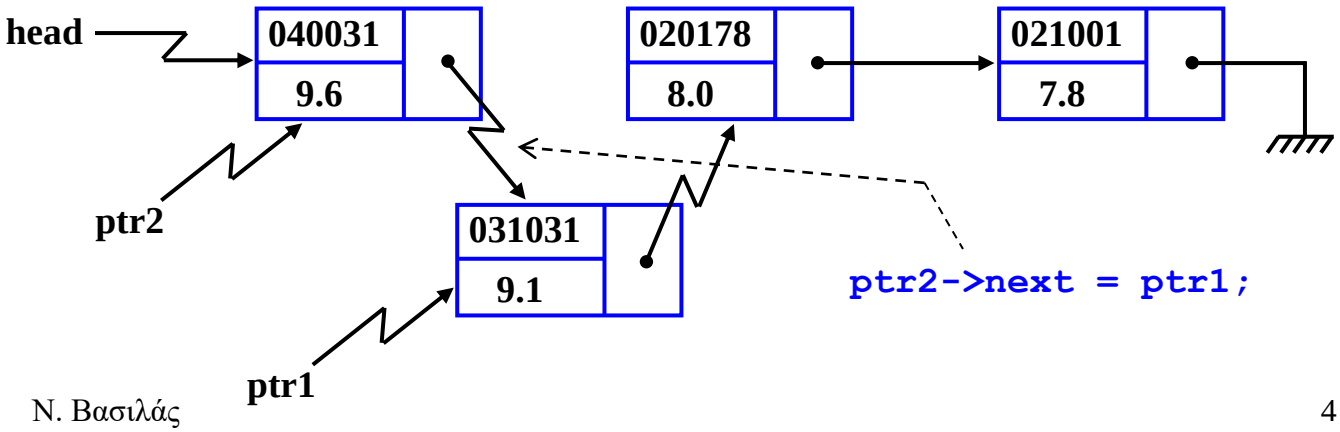
Παράδειγμα εισαγωγής κόμβου σε λίστα (μεταξύ 1ου και 2ου κόμβου)



Πρώτα τοποθετούμε έναν pointer (π.χ. τον ptr2) στον προηγούμενο κόμβο (στο παράδειγμά μας αυτός είναι ο πρώτος κόμβος) και συνδέουμε τον νέο κόμβο (στον οποίο δείχνει ο ptr1) με τον επόμενο κόμβο.

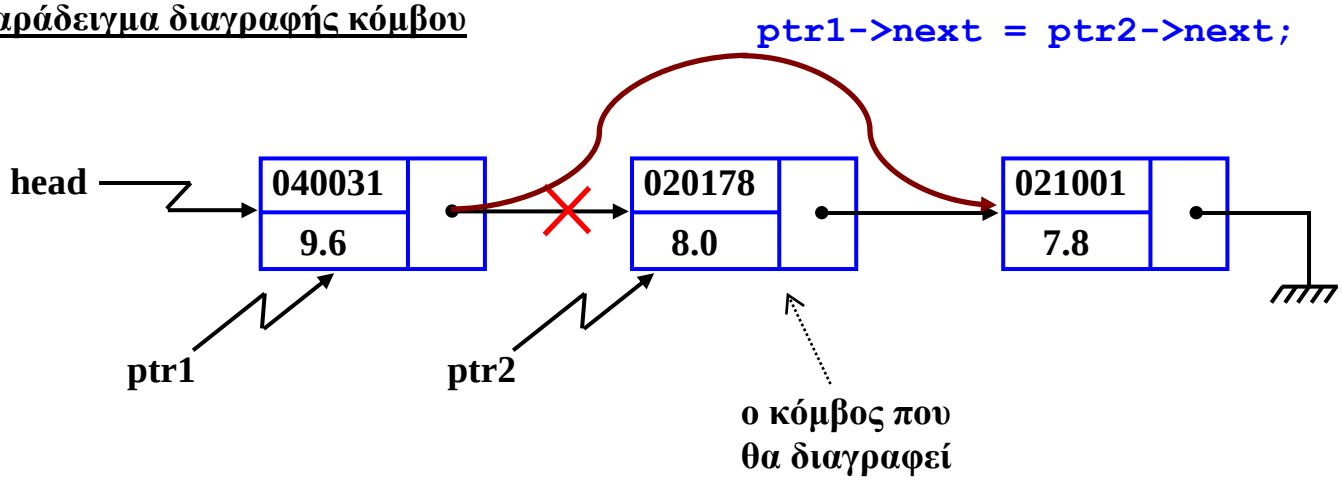


Τέλος, συνδέουμε τον προηγούμενο κόμβο με τον νέο:



```
/* Συνάρτηση που δέχεται μια λίστα και τα στοιχεία ενός κόμβου,
   εισάγει νέο κόμβο διατηρώντας τη φθίνουσα διάταξη των κόμβων
   ως προς το πεδίο του βαθμού και επιστρέφει τη νέα λίστα */
struct node *insert_node(h,am,bathmos)
struct node *h;
char *am;
float bathmos;
{
  struct node *p1,*p2;

  p2 = (struct node *) malloc(sizeof(struct node));
  strcpy(p2->AM,am);
  p2->grade = bathmos;
  p1 = h;  /* Ο p1 δείχνει στην αρχή της λίστας */
  if(p1 == NULL || p1->grade <= bathmos)
  {
    /* Εισαγωγή στην αρχή της λίστας */
    p2->next = p1;
    h = p2;
  }
  else
  {
    while(p1->next != NULL && p1->next->grade > bathmos)
      p1 = p1->next;
    if(p1->next == NULL) /* Πρόσθεσε στο τέλος της λίστας */
      { p1->next = p2;   p2->next = NULL; }
    else
    {
      p2->next = p1->next;
      p1->next = p2;
    }
  }
  return(h);
}
```

Παράδειγμα διαγραφής κόμβου



Τοποθετούμε έναν pointer (π.χ. τον ptr1) στον προηγούμενο κόμβο από αυτόν που πρόκειται να διαγραφεί και έναν (π.χ. τον ptr2) στον κόμβο που θα διαγραφεί. Στη συνέχεια, κάνουμε τον κόμβο του ptr1 να δείχνει στον μεθεπόμενο κόμβο. Για παράδειγμα, αν θέλουμε να διαγράψουμε τον κόμβο με ΑΜ = "020178":

```
struct node *head,*ptr1,*ptr2;
...
ptr1 = head;
if (strcmp(ptr1->AM,"020178") == 0) /*Αν είναι ο πρώτος κόμβος*/
{
  head = head->next;
  free(ptr1);  /* Αποδεσμεύει τη μνήμη για τον πρώτο κόμβο */
}
else   /* Δεν ήταν ο πρώτος κόμβος */
{
  while(ptr1->next != NULL && strcmp(ptr1->next->AM,"020178"))
    ptr1 = ptr1->next;
  if(ptr1->next != NULL)
  {
    ptr2 = ptr1->next;
    ptr1->next = ptr2->next;
    free(ptr2);  /* Αποδέσμευση μνήμης κόμβου που διαγράφηκε */
  }
}
```