ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Δυναμική Δέσμευση Μνήμης και Δομές Δεδομένων (Φροντιστήριο)

Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

http://www.cs.ucy.ac.cy/courses/EPL232

Dr. Andreas Aristidou a.aristidou@ieee.org

Προσθήκη κόμβου στην αρχή μιας συνδεδεμένης λίστας

- Ένα από τα πλεονεκτήματα μιας συνδεδεμένης λίστας είναι ότι οι κόμβοι μπορούν να προστεθούν σε οποιοδήποτε σημείο της λίστας.
- Η αρχή μιας λίστας είναι η ευκολότερη θέση για την εισαγωγή ενός κόμβου.
- Ας υποθέσουμε ότι η `new_node` δείχνει στον κόμβο που θα εισαχθεί, και η `first` δείχνει στον πρώτο κόμβο της συνδεδεμένης λίστας.

Dr. Andreas Aristidou a.aristidou@ieee.org

Προσθήκη κόμβου στην αρχή μιας συνδεδεμένης λίστας

- Χρειάζονται δύο δηλώσεις για να εισαχθεί ο κόμβος στη λίστα.
- Το πρώτο βήμα είναι να κάνουμε το νέο στοιχείο του κόμβου `next` να δείχνει στον κόμβο που προηγουμένως έδειχνε η αρχή της λίστας:

```
new_node->next = first;
```

- Το δεύτερο βήμα είναι να κάνουμε το `first` να δείχνει στο νέο κόμβο:

```
first = new_node;
```

- Αυτές οι δηλώσεις λειτουργούν ακόμα και αν η λίστα είναι κενή.

Dr. Andreas Aristidou a.aristidou@ieee.org

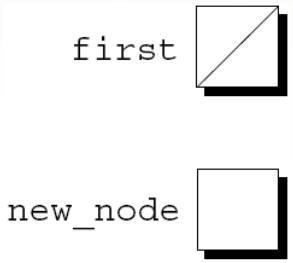
Προσθήκη κόμβου στην αρχή μιας συνδεδεμένης λίστας

- Ας δούμε τη διαδικασία εισαγωγής δύο κόμβων σε μια κενή λίστα.
- Θα εισάγουμε έναν κόμβο που περιέχει τον αριθμό 10, και μετά έναν κόμβο που περιέχει τον αριθμό 20.

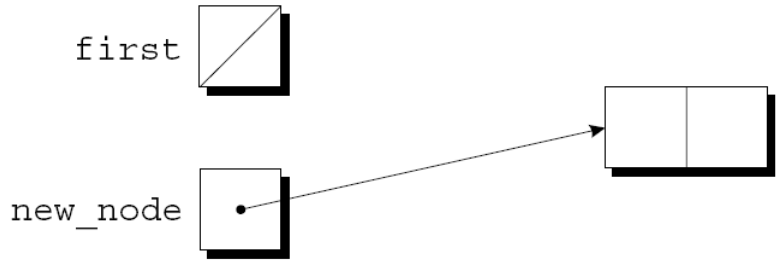
Dr. Andreas Aristidou a.aristidou@ieee.org

Προσθήκη κόμβου στην αρχή μιας συνδεδεμένης λίστας

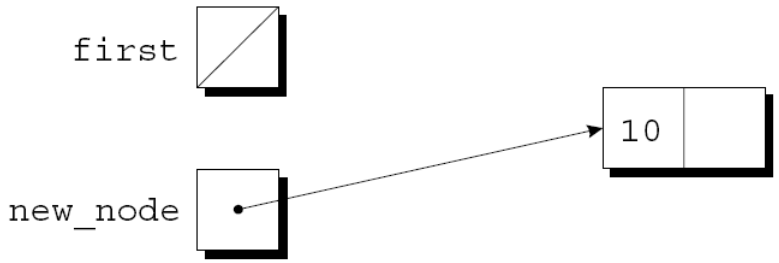
```
first = NULL;
```




```
new_node =
  malloc(sizeof(struct node));
```



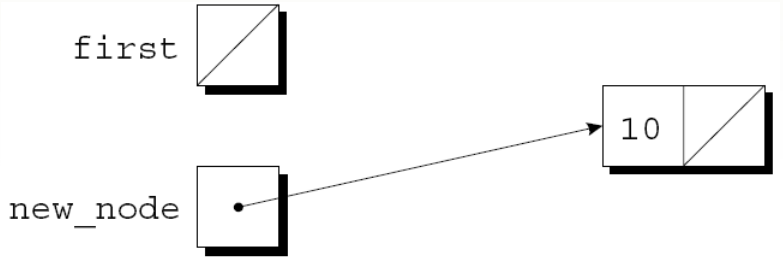

```
new_node->value = 10;
```



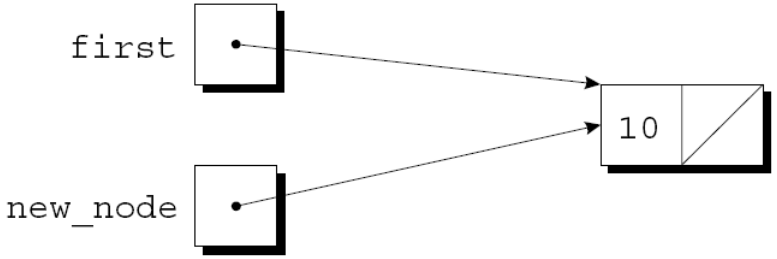
Dr. Andreas Aristidou a.aristidou@ieee.org

Προσθήκη κόμβου στην αρχή μιας συνδεδεμένης λίστας

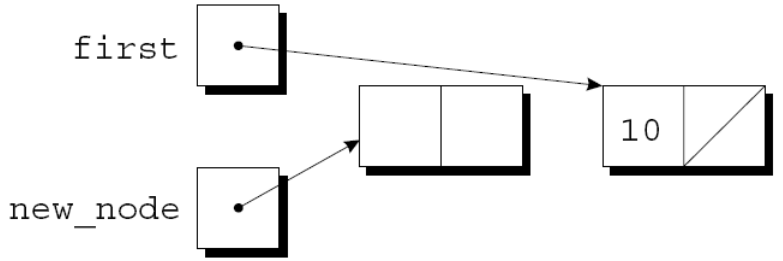
```
new_node->next = first;
```




```
first = new_node;
```

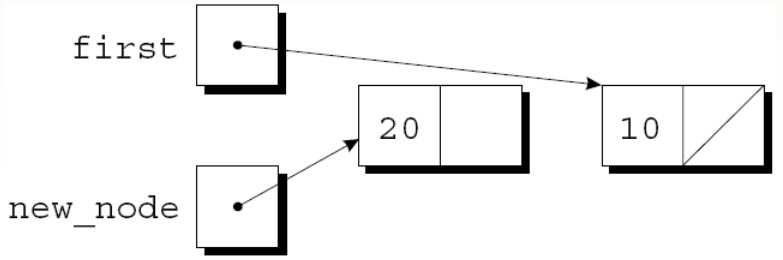



```
new_node =
  malloc(sizeof(struct node));
```

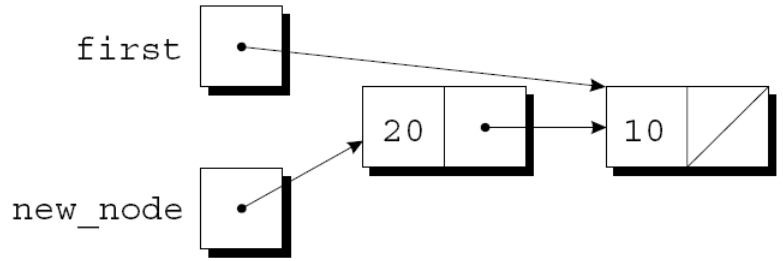


Προσθήκη κόμβου στην αρχή μιας συνδεδεμένης λίστας

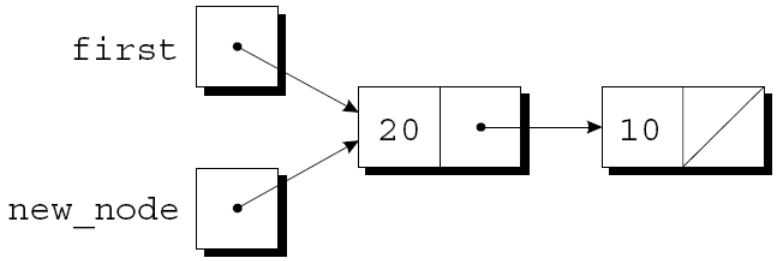
```
new_node->value = 20;
```




```
new_node->next = first;
```




```
first = new_node;
```



Dr. Andreas Aristidou a.aristidou@ieee.org

Προσθήκη κόμβου στην αρχή μιας συνδεδεμένης λίστας

- Μια συνάρτηση που εισάγει έναν κόμβο που περιέχει τον αριθμό `n` σε μια συνδεδεμένη λίστα, η οποία δείχνετε από το `list`:

```
struct node *add_to_list(struct node *list, int n)
{
  struct node *new_node;

  new_node = malloc(sizeof(struct node));
  if (new_node == NULL) {
    printf("Error: malloc failed in add_to_list\n");
    exit(EXIT_FAILURE);
  }
  new_node->value = n;
  new_node->next = list;
  return new_node;
}
```

Προσθήκη κόμβου στην αρχή μιας συνδεδεμένης λίστας

- Σημειώστε ότι η `add_to_list` επιστρέφει ένα δείκτη στον κόμβο που μόλις δημιουργήσατε.
- Όταν καλέσουμε την `add_to_list`, θα πρέπει να αποθηκεύσουμε την επιστρεφόμενη τιμή στο `first`:

```
first = add_to_list(first, 10);
first = add_to_list(first, 20);
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Προσθήκη κόμβου στην αρχή μιας συνδεδεμένης λίστας

- Μια συνάρτηση που χρησιμοποιεί την `add_to_list` για να δημιουργήσει μια συνδεδεμένη λίστα που περιέχει αριθμούς που έχουν εισαχθεί από το χρήστη:

```
struct node *read_numbers(void)
{
  struct node *first = NULL;
  int n;

  printf("Enter a series of integers (0 to terminate): ");
  for (;;) {
    scanf("%d", &n);
    if (n == 0)
      return first;
    first = add_to_list(first, n);
  }
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Αναζήτηση σε μια συνδεδεμένη λίστα

- Αν και ο βρόγχος `while` μπορεί να χρησιμοποιηθεί για την αναζήτηση σε μια λίστα, ο βρόγχος `for` είναι συχνά ανώτερος.
- Ένας βρόχος που επισκέπτεται τους κόμβους μιας συνδεδεμένης λίστας, χρησιμοποιώντας μια μεταβλητή δείκτη `p` για να παρακολουθεί τον "τρέχοντα" κόμβο είναι:

```
for (p = first; p != NULL; p = p->next)
   …
```

- Ένας βρόχος αυτής της μορφής μπορεί να χρησιμοποιηθεί σε μια συνάρτηση που αναζητά έναν ακέραιο αριθμό `n`.

Dr. Andreas Aristidou a.aristidou@ieee.org

Αναζήτηση σε μια συνδεδεμένη λίστα

- Εάν βρει το `n`, η λειτουργία θα επιστρέψει ένα δείκτη στον κόμβο που περιέχει το `n`; διαφορετικά, θα επιστρέψει ένα μηδενικό δείκτη.
- Μια αρχική έκδοση της συνάρτησης είναι:

```
struct node *search_list(struct node *list, int n)
{
  struct node *p;

  for (p = list; p != NULL; p = p->next)
    if (p->value == n)
      return p;
  return NULL;
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Αναζήτηση σε μια συνδεδεμένη λίστα

- Μια εναλλακτική λύση είναι η εξάλειψη της μεταβλητής `p`, και χρήση της `list` για την παρακολούθηση του τρέχοντα κόμβου:

```
struct node *search_list(struct node *list, int n)
{
  for (; list != NULL; list = list->next)
    if (list->value == n)
      return list;
  return NULL;
}
```

- Αφού η `list` είναι ένα αντίγραφο του αρχικού δείκτη της λίστας, δεν βλάπτει να το αλλάξετε μέσα στη λειτουργία.

Dr. Andreas Aristidou a.aristidou@ieee.org

Αναζήτηση σε μια συνδεδεμένη λίστα

- Εναλλακτικά, με χρήση του `while`:

```
struct node *search_list(struct node *list, int n)
{
  while (list != NULL && list->value != n)
    list = list->next;
  return list;
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Διαγραφή κόμβου σε συνδεδεμένη λίστα

- Ένα μεγάλο πλεονέκτημα της αποθήκευσης δεδομένων σε μια συνδεδεμένη λίστα είναι ότι μπορούμε εύκολα να διαγράψετε κόμβους.
- Η διαγραφή ενός κόμβου περιλαμβάνει τρία βήματα:
  1. Εντοπισμός του κόμβου που θα διαγραφεί.
  2. Τροποποίηση του προηγούμενου κόμβου, ώστε να "παρακάμπτει" τον διαγραμμένο κόμβο.
  3. Κλήση της `free` για να ανακτήσετε το χώρο που καταλαμβάνει ο διαγραμμένος κόμβος.
- Το Βήμα 1 είναι δυσκολότερο από ό, τι φαίνεται, επειδή το βήμα 2 απαιτεί την αλλαγή του προηγούμενου κόμβου.
- Υπάρχουν διάφορες λύσεις σε αυτό το πρόβλημα.

Dr. Andreas Aristidou a.aristidou@ieee.org

Διαγραφή κόμβου σε συνδεδεμένη λίστα

- Η τεχνική "τελικού δείκτη" περιλαμβάνει τη διατήρηση ενός δείκτη στον προηγούμενο κόμβο (`prev`) καθώς και ένα δείκτη στον τρέχοντα κόμβο (`cur`).
- Ας υποθέσουμε ότι η `list` δείχνεθ στη λίστα προς αναζήτηση και `n` είναι ο ακέραιος που θα διαγραφεί.
- Ένας βρόγχος που υλοποιεί το βήμα 1:

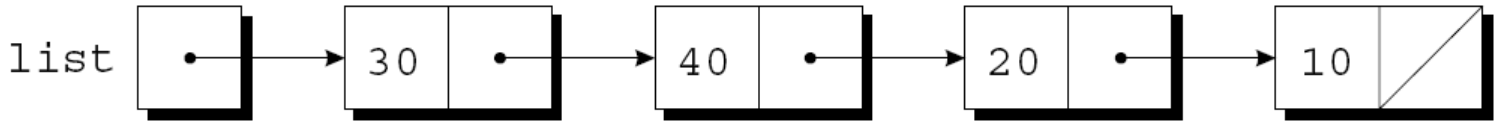
```
for (cur = list, prev = NULL;
     cur != NULL && cur->value != n;
     prev = cur, cur = cur->next) ;
```

- Όταν ο βρόγχος τερματίζεται, η `cur` δείχνει στον κόμβο που θα διαγραφεί και η `prev` δείχνει στον προηγούμενο κόμβο.

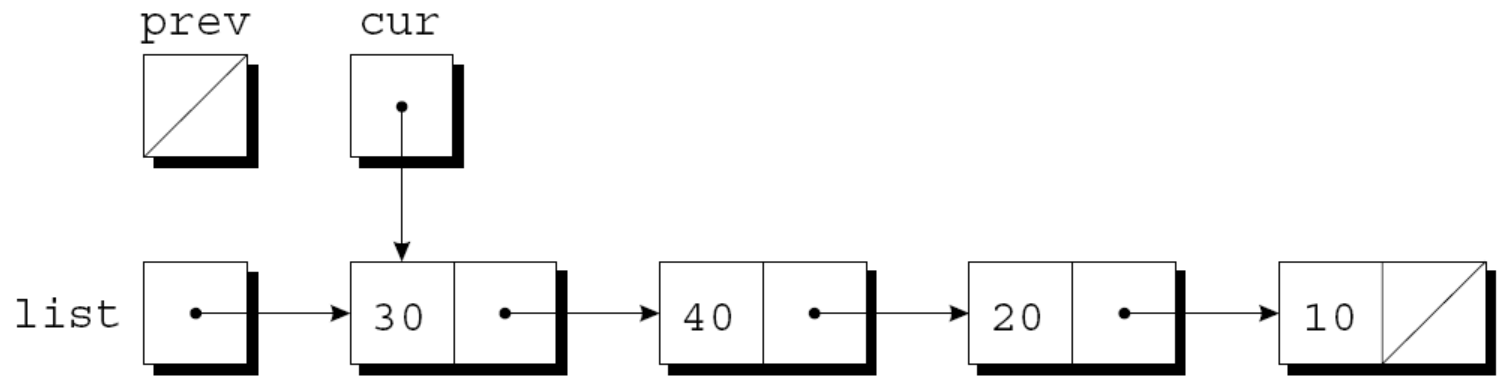
Dr. Andreas Aristidou a.aristidou@ieee.org

Διαγραφή κόμβου σε συνδεδεμένη λίστα

- Ας υποθέσουμε ότι η `list` έχει την ακόλουθη εμφάνιση και `n` είναι 20:

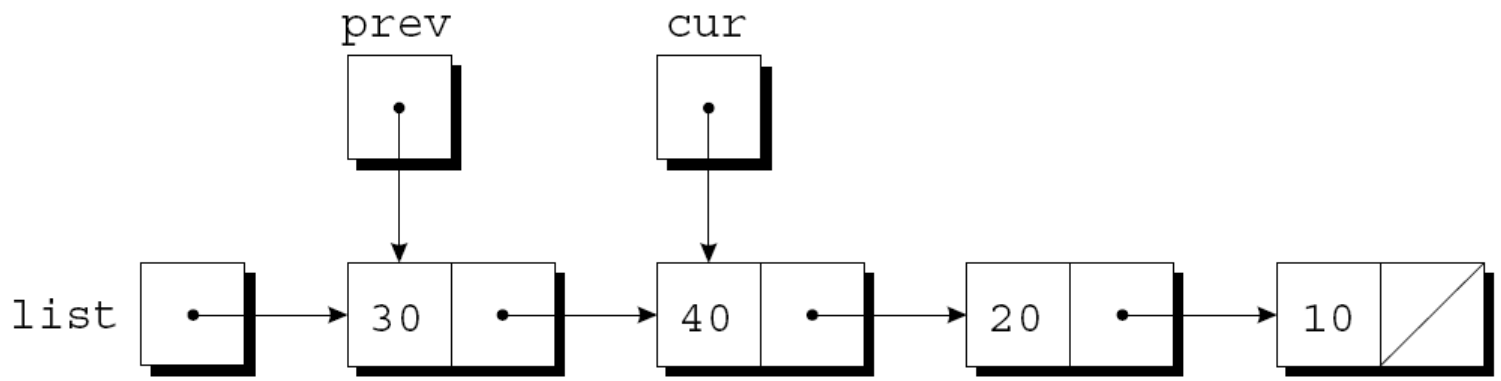


- Ακολούθως, η `cur = list,` και `prev = NULL` έχουν εκτελεστεί:



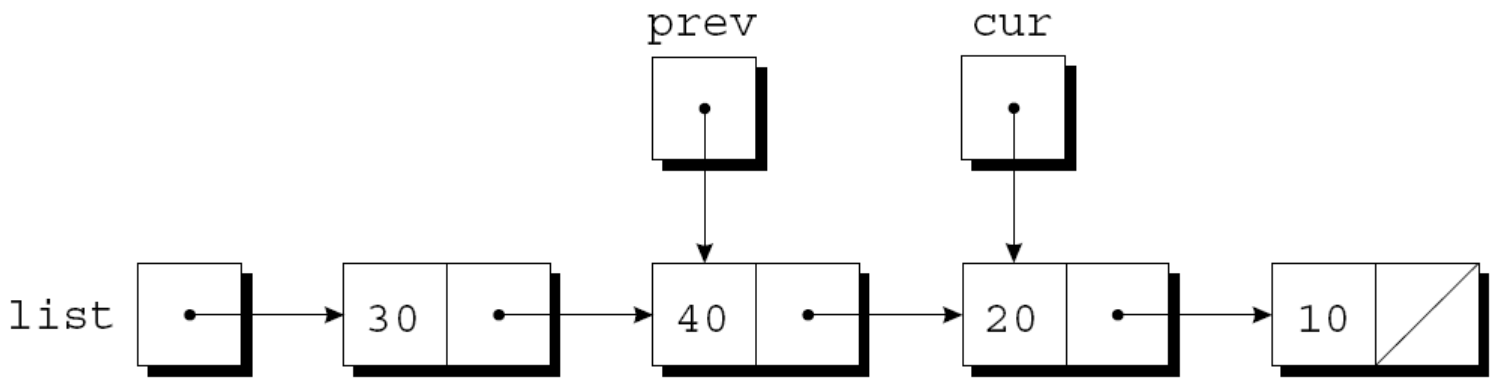
Διαγραφή κόμβου σε συνδεδεμένη λίστα

- Ο έλεγχος `cur != NULL && cur->value != n` είναι ορθός, αφού η `cur` δείχνει σε έναν κόμβο και ο κόμβος δεν περιέχει το 20.
- Ακολούθως, η `prev = cur, cur = cur->next` έχουν εκτελεστεί:



Διαγραφή κόμβου σε συνδεδεμένη λίστα

- Ο έλεγχος `cur != NULL && cur->value != n` είναι πάλι σωστός, οπότε `prev = cur, cur = cur->next` εκτελείται ξανά:



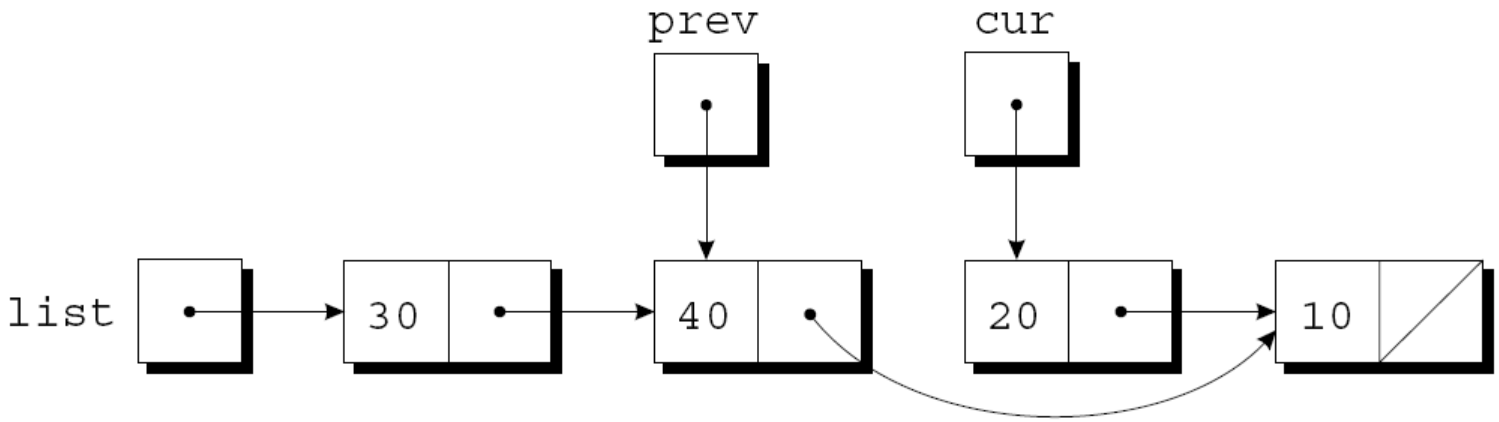
- Αφού το `cur` τώρα δείχνει στον κόμβο που περιέχει το 20, ο έλεγχος `cur->value != n` είναι λάθος και ο βρόγχος τερματίζει.

Διαγραφή κόμβου σε συνδεδεμένη λίστα

- Στη συνέχεια, θα εκτελέσουμε την παράκαμψη που απαιτείται από το βήμα 2.
- Η δήλωση

```
prev->next = cur->next;
```

κάνει το δείκτη στο προηγούμενο κόμβο να δείχνει στον κόμβο που δίχνει η τρέχουσα τιμή του κόμβου:



Διαγραφή κόμβου σε συνδεδεμένη λίστα

- Το Βήμα 3 είναι να απελευθερώσει τη μνήμη που καταλαμβάνει ο τρέχων κόμβος:

```
free(cur);
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Διαγραφή κόμβου σε συνδεδεμένη λίστα

- Η συνάρτηση `delete_from_list` χρησιμοποιεί τη στρατηγική που μόλις εξηγήσαμε.
- Όταν δίνεται μια λίστα και ένας ακέραιος `n`, η συνάρτηση διαγράφει τον πρώτο κόμβο που περιέχει τον αριθμό `n`.
- Εάν δεν υπάρχει κόμβος που να περιέχει το `n`, η `delete_from_list` δεν κάνει τίποτα.
- Η διαγραφή του πρώτου κόμβου στη λίστα είναι μια ειδική περίπτωση που απαιτεί διαφορετικό βήμα παράκαμψης.

Διαγραφή κόμβου σε συνδεδεμένη λίστα

```
struct node *delete_from_list(struct node *list, int n)
{
  struct node *cur, *prev;

  for (cur = list, prev = NULL;
       cur != NULL && cur->value != n;
       prev = cur, cur = cur->next)
    ;
  if (cur == NULL)
    return list;                  /* n was not found */
  if (prev == NULL)
    list = list->next;            /* n is in the first node */
  else
    prev->next = cur->next;       /* n is in some other node */
  free(cur);
  return list;
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Ασκήσεις κατανόησης

- Γράψτε την συνάρτηση `int *create_array`, η οποία επιστρέφει ένα δείκτη σε ένα δυναμικά δεσμευμένο πίνακα τύπου `int` με `n` στοιχεία, τα οποία αρχικοποιούνται σε μια `initial_value`. Η συνάρτηση θα πρέπει να επιστρέφει NULL εάν ο πίνακας δεν μπορεί να δεσμευτεί.

```
int *create_array(int n, int initial_value)
{
  int *a, *p;

  a = malloc(n * sizeof(int));
  if (a != NULL)
    for (p = a; p < a + n; p++)
      *p = initial_value;
  return a;
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org

Ασκήσεις κατανόησης

- Έστω οι ακόλουθες δηλώσεις:

```
struct point{ int x, y; }
struct rectangle{struct point upper_left, lower right; }
struct rectangle *p
```

Έστω πως ο δείκτης `p` δείχνει στη δομή `rectangle` όπου η άνω αριστερή γωνία είναι στο σημείο (10,25), και η κάτω δεξιά γωνία είναι στο (20,15). Γράψτε μια σειρά από δηλώσεις που δεσμεύουν μια τέτοια δομή και την αρχικοποιούν αναλογα.

```
p = malloc(sizeof(struct rectangle));
p->upper_left.x = 10;
p->upper_left.y = 25;
p->lower_right.x = 20;
p->lower_right.y = 15;
```


Dr. Andreas Aristidou a.aristidou@ieee.org

Ασκήσεις κατανόησης

- Ο ακόλουθος βρόγχος υποτίθεται πως διαγράφει όλους τους κόμβους από μια συνδεδεμένη λίστα και ελευθερώνει την μνήμη που καταλαμβάνουν. Δυστυχώς είναι λάθος. Εξηγήστε γιατί και διορθώστε.

```
for (p = first; p != NULL; p = p->next)
    free(p);
```

Πρέπει να έχουμε τον επόμενο δείκτη πρωτού ελευθερώσουμε την μνήμη για τον κόμβο:

```
struct node *p:
while((p = list) != NULL)    // set p to list, stop if empty.
{
    list = list- > next;   // advance list to next element
    free(p);               // delete saved pointer.
}
```

Dr. Andreas Aristidou a.aristidou@ieee.org