

Συνδεδεμένες λίστες

#17

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Πανεπιστήμιο Ιωαννίνων (Άρτα)

Γκόγκος Χρήστος

Δομές δεδομένων για την αποθήκευση μιας συλλογής στοιχείων

- Συχνά υπάρχει η ανάγκη χρήσης δομών δεδομένων για την αποθήκευση μιας συλλογής στοιχείων.
 - Τυπικές λειτουργίες σε αυτές τις δομές είναι η εισαγωγή, η διαγραφή, η εύρεση μέγιστου στοιχείου, η ενημέρωση κ.λπ.

Δομές δεδομένων για την αποθήκευση μιας συλλογής αντικειμένων

- Τι επιλογές έχουμε;
 - **Στατικοί πίνακες**
 - Περιορισμοί:
 - Προκαθορισμένη χωρητικότητα, άρα μπορεί να μη γίνεται καλή χρήση του διαθέσιμου χώρου που έχει δεσμευθεί στη μνήμη.
 - Οι λειτουργίες εισαγωγής και διαγραφής μπορεί να έχουν υψηλό κόστος (καθώς γίνονται πολλές αντιγραφές) αν δεν θέλουμε να αφήνουμε κενά ενδιάμεσα στον πίνακα.
 - Πλεονεκτήματα:
 - Ευκολία προγραμματισμού.
 - Συνεχόμενες θέσεις μνήμης για εύκολη δεικτοδότηση.
 - **Δυναμικοί πίνακες**
 - Περιορισμοί:
 - Η χωρητικότητα είναι δυναμική, η μνήμη πάλι μπορεί να μη χρησιμοποιείται με τον καλύτερο τρόπο, αλλά σε ορισμένες περιπτώσεις αποτελεί καλύτερη λύση σε σχέση με τους στατικούς πίνακες.
 - Οι λειτουργίες εισαγωγής και διαγραφής μπορεί να έχουν υψηλό κόστος, ειδικά όταν η χωρητικότητα αλλάζει.
 - Πλεονεκτήματα:
 - Ευκολία προγραμματισμού (ο προγραμματιστής ωστόσο πρέπει να φροντίζει για τη δέσμευση και την αποδέσμευση μνήμης).
 - Συνεχόμενες θέσεις μνήμης για εύκολη δεικτοδότηση.

Δομές δεδομένων για την αποθήκευση μιας συλλογής αντικειμένων

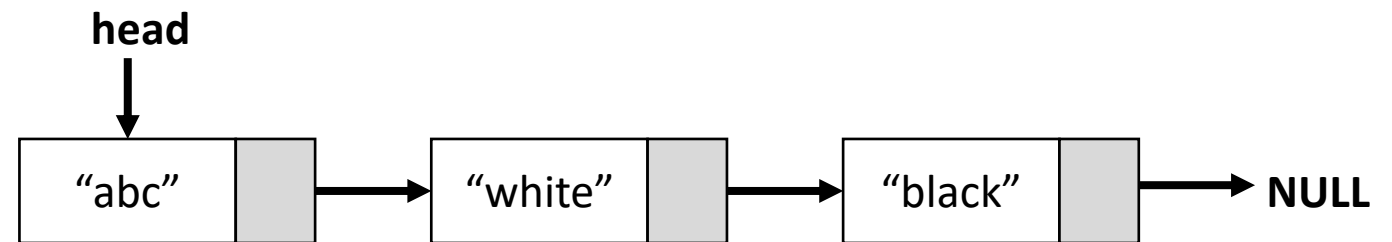
- Μια ακόμα λύση είναι οι συνδεδεμένες λίστες.
 - Πρόκειται για μια πραγματικά δυναμική δομή δεδομένων καθώς κάθε στοιχείο της δεσμεύεται δυναμικά χρησιμοποιώντας τον τελεστή new όταν αυτό χρειάζεται.
 - Η χωρητικότητα της συνδεδεμένης λίστας είναι πάντα ίση με τη μνήμη που έχει δεσμευθεί (συν κάποια επιβάρυνση για τους δείκτες που χρειάζονται).
 - Οι λειτουργίες εισαγωγής και διαγραφής έχουν χαμηλό κόστος.
 - Οι θέσεις μνήμης που καταλαμβάνει δεν είναι συνεχόμενες.
 - Σημαντικός περιορισμός είναι ότι είτε δεν ορίζεται ο τελεστής [], είτε έχει υψηλό κόστος χρόνου εκτέλεσης.
- Οι συνδεδεμένες λίστες είναι μια από τις «συνδεδεμένες δομές δεδομένων».
- Άλλες συνδεδεμένες δομές δεδομένων είναι τα δένδρα, τα γραφήματα κ.α.

Συνδεδεμένες λίστες και πίνακες

- Ένας πίνακας λεκτικών:
 - `string s[5];`
 - `s[0]="abc";`
 - `s[1]="white";`
 - `s[2]="black";`
- Μια συνδεδεμένη λίστα λεκτικών:
 - Κάθε στοιχείο της λίστας έχει δύο πεδία
 - Ένα πεδίο `string`.
 - Ένα δείκτη που δείχνει προς το επόμενο στοιχείο της λίστας.

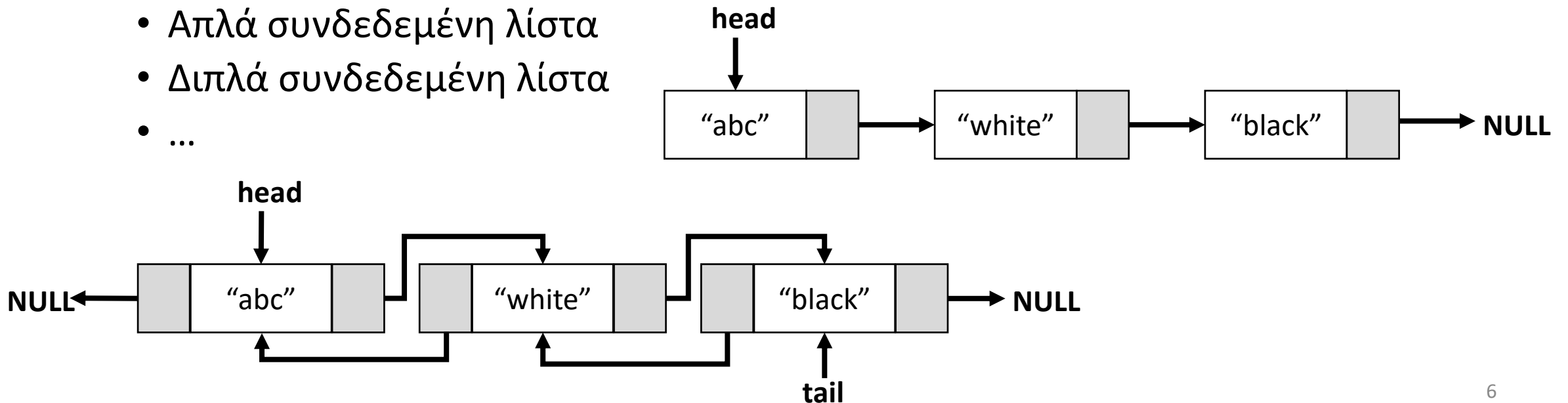
```
class listnode
{
public:
    string item;
    listnode *next;
};
```

s	
0	"abc"
1	"white"
2	"black"
3	
4	



Συνδεδεμένες λίστες

- Μειώνεται η σπατάλη μνήμης (απαίτηση επιπλέον χώρου μόνο για τους δείκτες).
- Κάθε κόμβος της λίστας δεσμεύεται δυναμικά με τον τελεστή new.
- Υπάρχουν πολλές παραλλαγές συνδεδεμένων λιστών:
 - Απλά συνδεδεμένη λίστα
 - Διπλά συνδεδεμένη λίστα
 - ...



Μια διπλά συνδεδεμένη λίστα

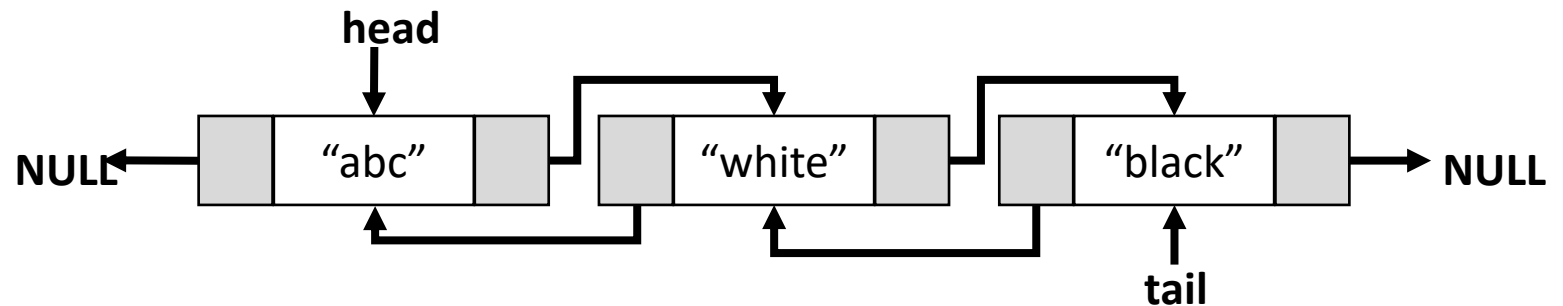
- Ας υποθέσουμε ότι αποθηκεύουμε δύο πεδία δεδομένων σε κάθε κόμβο (ένα λεκτικό και έναν ακέραιο). Τότε η δομή δεδομένων `listnode` θα είναι η ακόλουθη:

```
class listnode
{
    public:
        string s;
        int count;
        listnode *next;
        listnode *prev;
        listnode(): s(""), count(0), next(NULL), prev(NULL) {};
        listnode(const string& ss, const int &c): s(ss), count(c), next(NULL), prev(NULL) {};
};
```

Τα ιδιωτικά δεδομένα της διπλά συνδεδεμένης λίστας

- Δεδομένα που προστατεύονται:
 - `head`: δείκτης προς την κεφαλή της λίστας για τον οποίο ισχύει ότι `head->prev == NULL`
 - `tail`: δείκτης προς το τελευταίο στοιχείο της λίστας για το οποίο ισχύει ότι `tail->next == NULL`
 - `size`: αριθμός των κόμβων της λίστας

```
class mylist {  
...  
private:  
    listnode* head;  
    listnode* tail;  
    int size;  
};
```



Η δημόσια διεπαφή (public interface) της mylist (1/2)

- `mylist();`
- `~mylist();`
- `void print() const;`
- `mylist(const mylist& l);`
- `mylist& operator=(const mylist &l);`
- `void insertfront(const string &s, const int &c);`
- `void insertback(const string &s, const int &c);`
- `void insertbefore(listnode *ptr, const string &s, const int &c);`
- `void insertafter(listnode *ptr, const string &s, const int &c);`
- `void insertpos(const int &pos, const string &s, const int &c);`

<https://github.com/chgogos/oop/blob/master/variuous/COP3330/lect17/mylist.h>

Η δημόσια διεπαφή (public interface) της mylist (2/2)

- `void removefront();`
- `void removeback();`
- `void remove(listnode* ptr);`
- `void removepos(const int& pos);`
- `listnode front() const;`
- `listnode back() const;`
- `int length() const;`
- `listnode* search(const string &s) const;`
- `listnode* findmaxcount() const;`
- `void removemaxcount();`
- `bool searchandinc(const string& s);`

<https://github.com/chgogos/oop/blob/master/various/COP3330/lect17/mylist.h>

Υλοποίηση της mylist

- Κατασκευαστές
 - **Προκαθορισμένος κατασκευαστής:** Δημιουργία μιας άδειας λίστας.
 - **Κατασκευαστής αντιγραφής:** Πρέπει να διανύει την υπάρχουσα λίστα και να εισάγει έναν νέο κόμβο με τις ίδιες τιμές στη λίστα (με τη χρήση του `insertback`).
- Καταστροφές
 - Πρέπει να διανύει τη λίστα και να διαγράφει όλους τους κόμβους της (που έχουν δημιουργηθεί με τη χρήση του τελεστή `new`).
- Τελεστής ανάθεσης, `operator=`
 - Παρόμοια λογική με τον κατασκευαστή αντιγραφής.
- Η συνάρτηση `print`
 - Διανύει τη λίστα και εμφανίζει ένα προς ένα τους κόμβους της.
- Οι υπόλοιπες συναρτήσεις αποτελούν διαφορετικές εκδόσεις της εισαγωγής (`insert`), διαγραφής (`remove`) και αναζήτησης (`search`) που θα παρουσιαστούν στη συνέχεια.

Εισαγωγή (insert)

- `insertback`

- Δύο περιπτώσεις:

- Εισαγωγή σε άδεια λίστα.
 - Εισαγωγή σε λίστα με στοιχεία.

- Εισαγωγή σε άδεια λίστα

- Δημιουργία ενός νέου κόμβου (`prev=NULL`, `next=NULL`), και τόσο ο δείκτης `head` όσο και δείκτης `tail` πρέπει να δείχνουν στο νέο κόμβο.

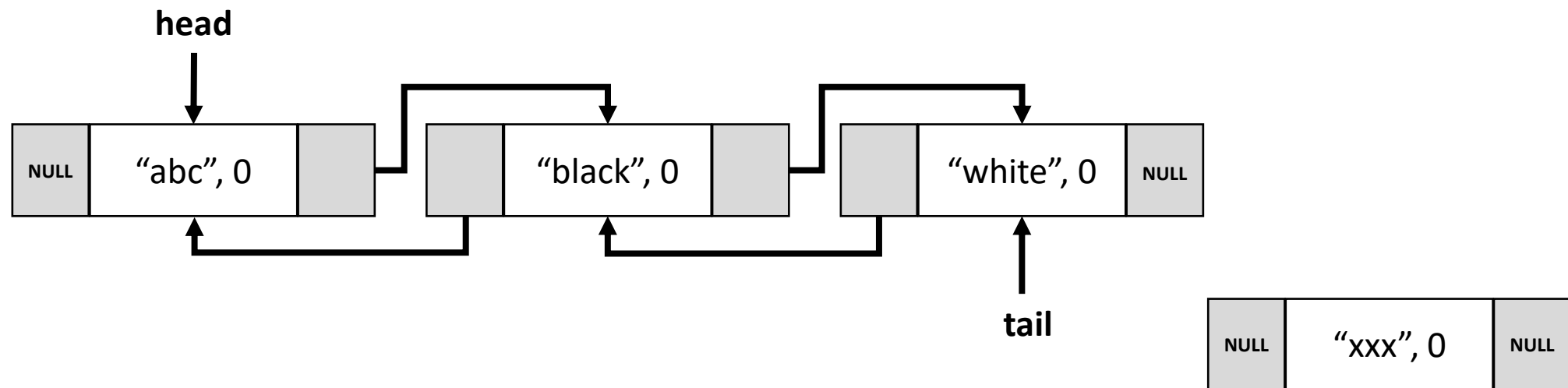
```
listnode *t = new listnode(s, c);  
if (head == NULL) {  
    head = t;  
    tail = t;  
    size++;  
}
```

insertback (1/4)

- insertback σε μια λίστα με στοιχεία

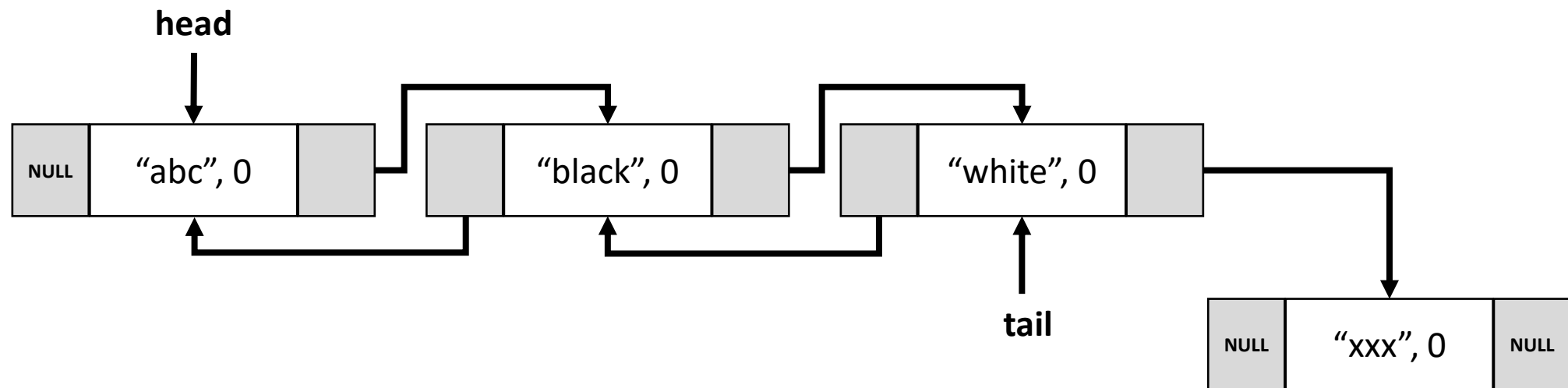
- Βήμα 1: Δημιουργία του νέου κόμβου

```
listnode *t = new listnode(s,c);
```



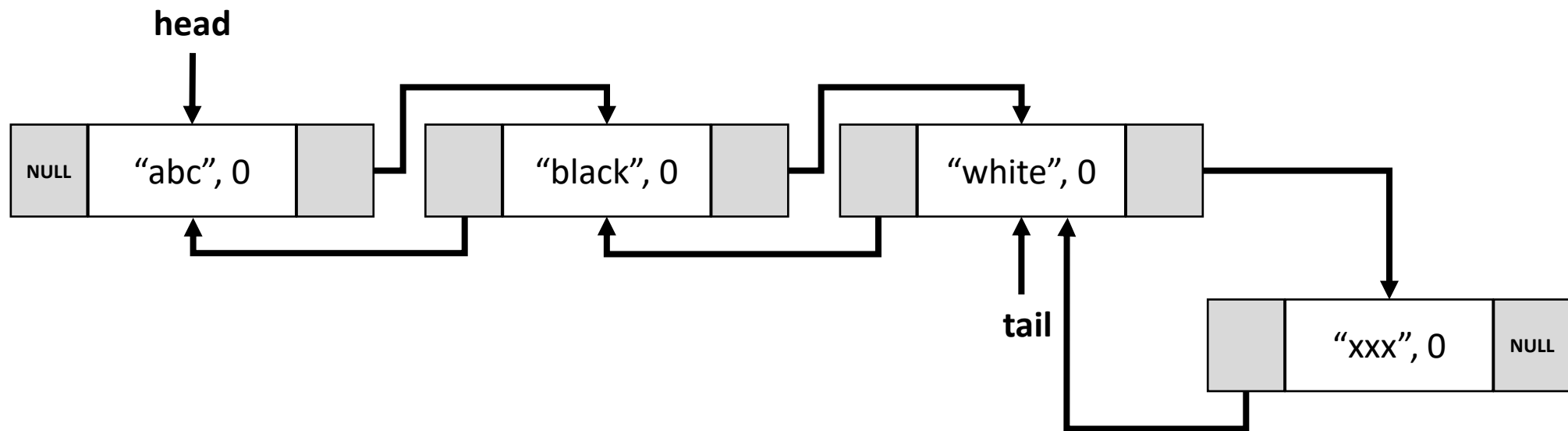
insertback (2/4)

- insertback σε μια λίστα με στοιχεία
 - Βήμα 2: σύνδεση του νέου κόμβου με την ουρά της λίστας (δείκτης next)
`tail->next = t;`



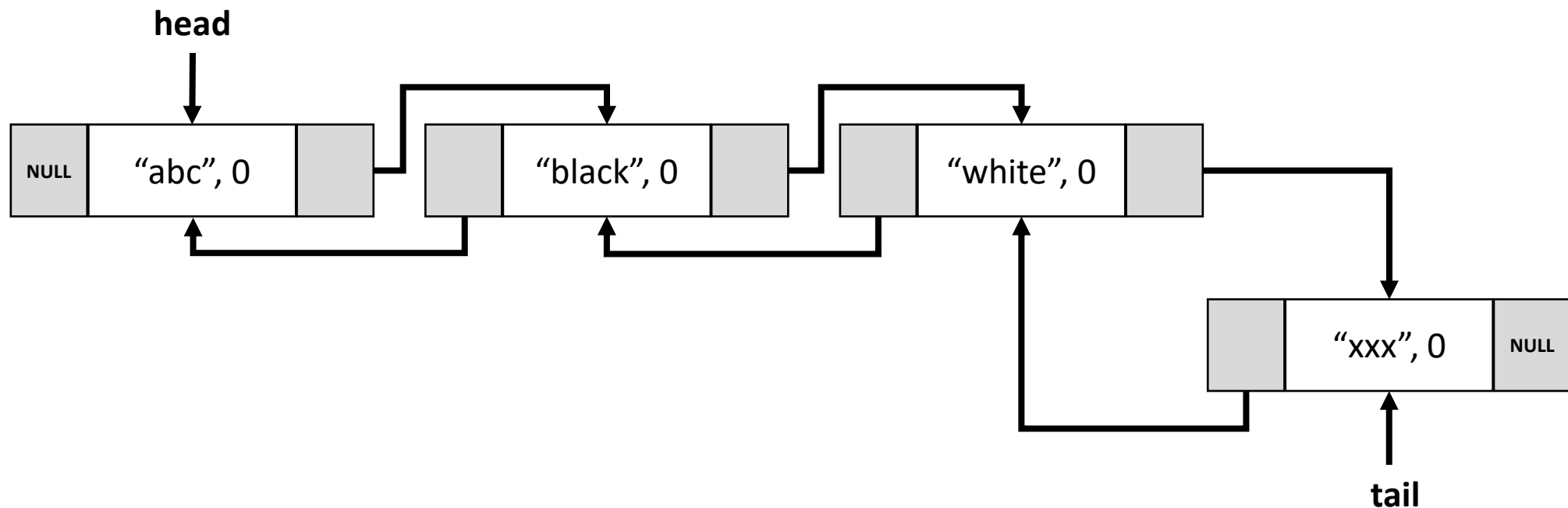
insertback (3/4)

- insertback σε μια λίστα με στοιχεία
 - Βήμα 3: σύνδεση του νέου κόμβου με την λίστα (δείκτης prev)
`t->prev = tail;`



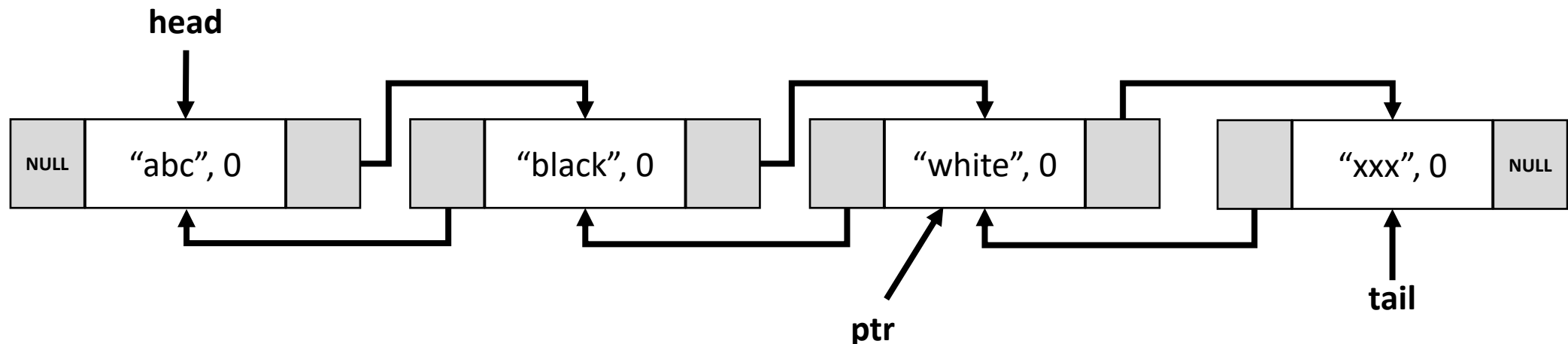
insertback (4/4)

- insertback σε μια λίστα με στοιχεία
 - Βήμα 4: αλλαγή του δείκτη tail έτσι ώστε να δείχνει στο νέο κόμβο
`tail = t;`



insertbefore (1/5)

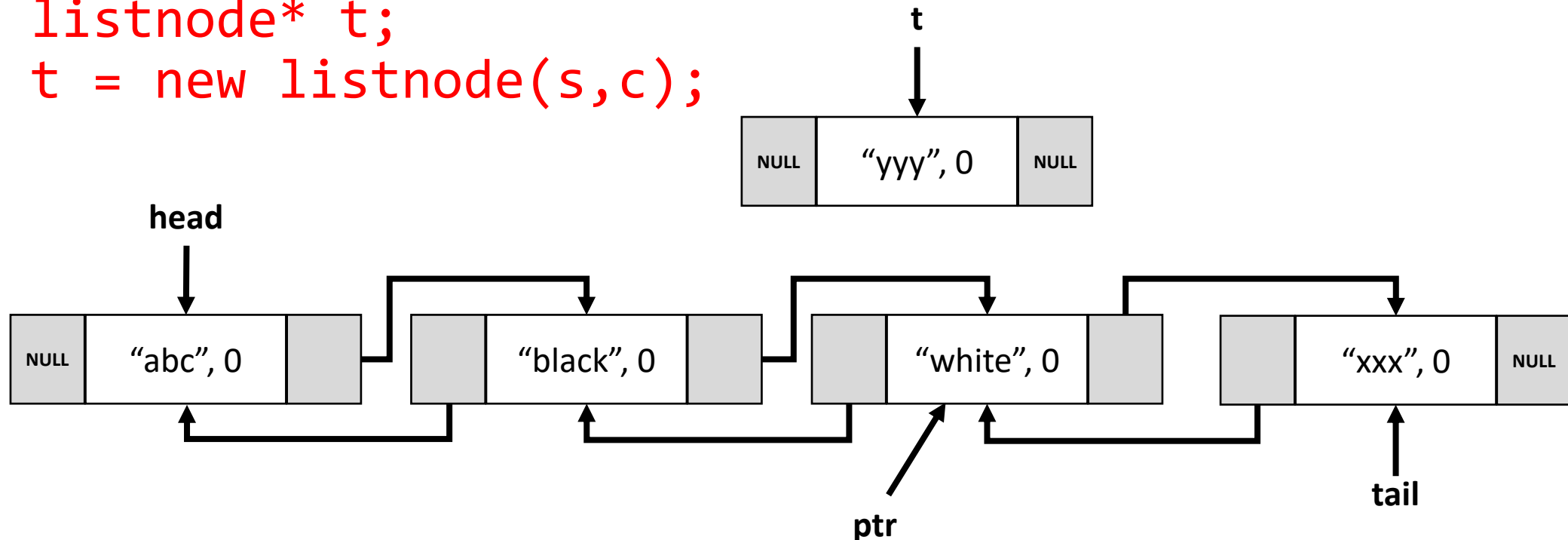
- Η εισαγωγή στην αρχή (`insertfront`) είναι παρόμοια με την `insertback`.
- Η εισαγωγή πριν τη κεφαλή είναι ισοδύναμη με την `insertfront`.
- Η εισαγωγή στο ενδιάμεσο της λίστας πριν το δείκτη `ptr` είναι διαφορετική: ένας νέος κόμβος πρόκειται να προστεθεί μεταξύ του `ptr->prev` και του `ptr`.



insertbefore (2/5)

- Η εισαγωγή στο ενδιάμεσο της λίστας πριν το δείκτη ptr.
 - Ένας νέος κόμβος πρόκειται να προστεθεί μεταξύ του ptr->prev και του ptr.
- Βήμα 1: δημιουργία ενός νέου κόμβου:

```
listnode* t;  
t = new listnode(s,c);
```

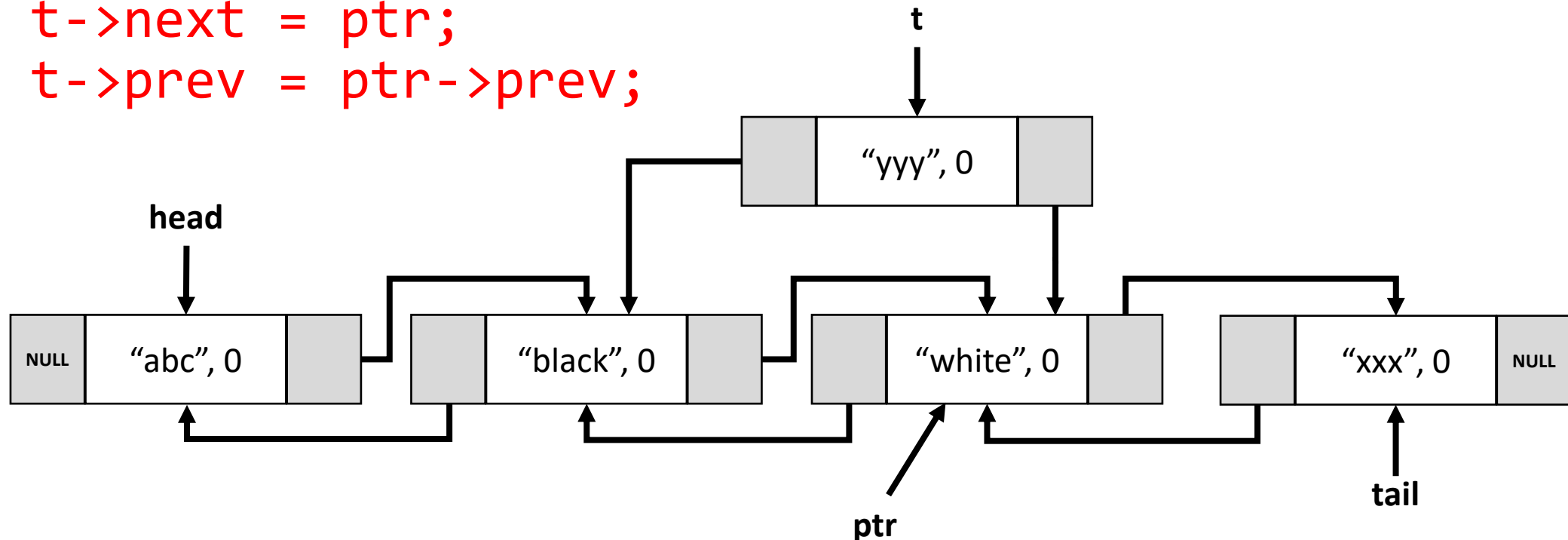


insertbefore (3/5)

- Η εισαγωγή στο ενδιάμεσο της λίστας πριν το δείκτη ptr.
 - Ένας νέος κόμβος πρόκειται να προστεθεί μεταξύ του ptr->prev και του ptr.
- Βήμα 2: σύνδεση του νέου κόμβου με τη λίστα:

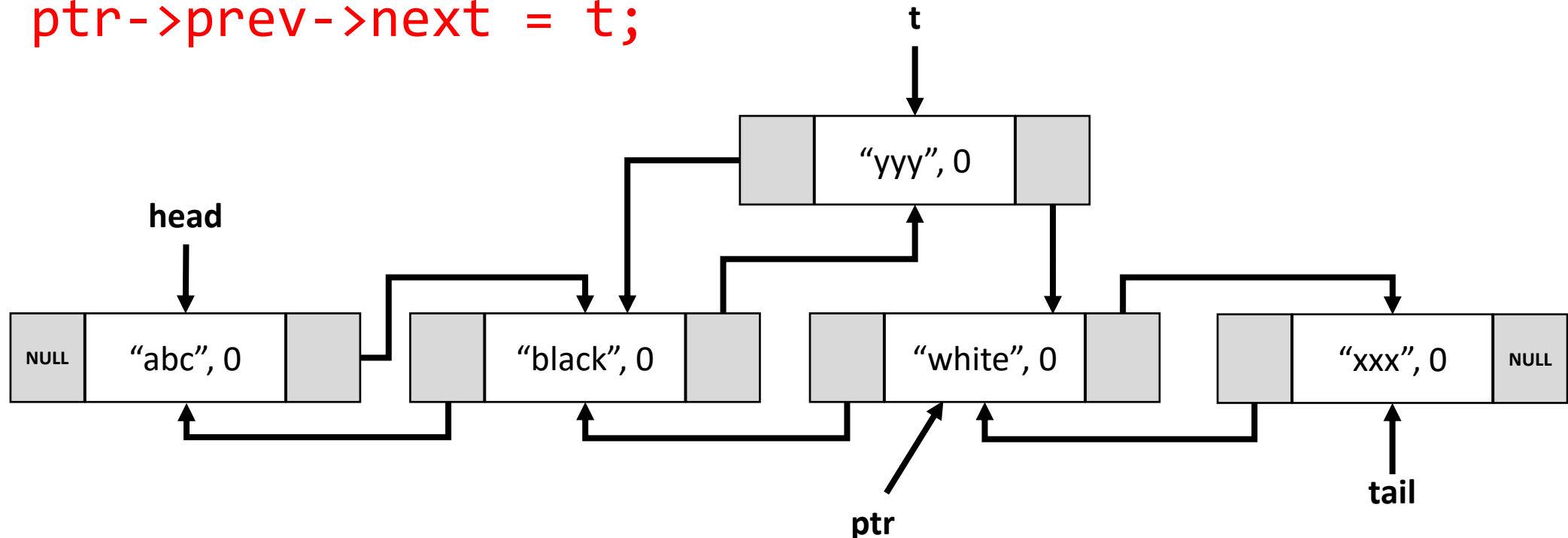
`t->next = ptr;`

`t->prev = ptr->prev;`



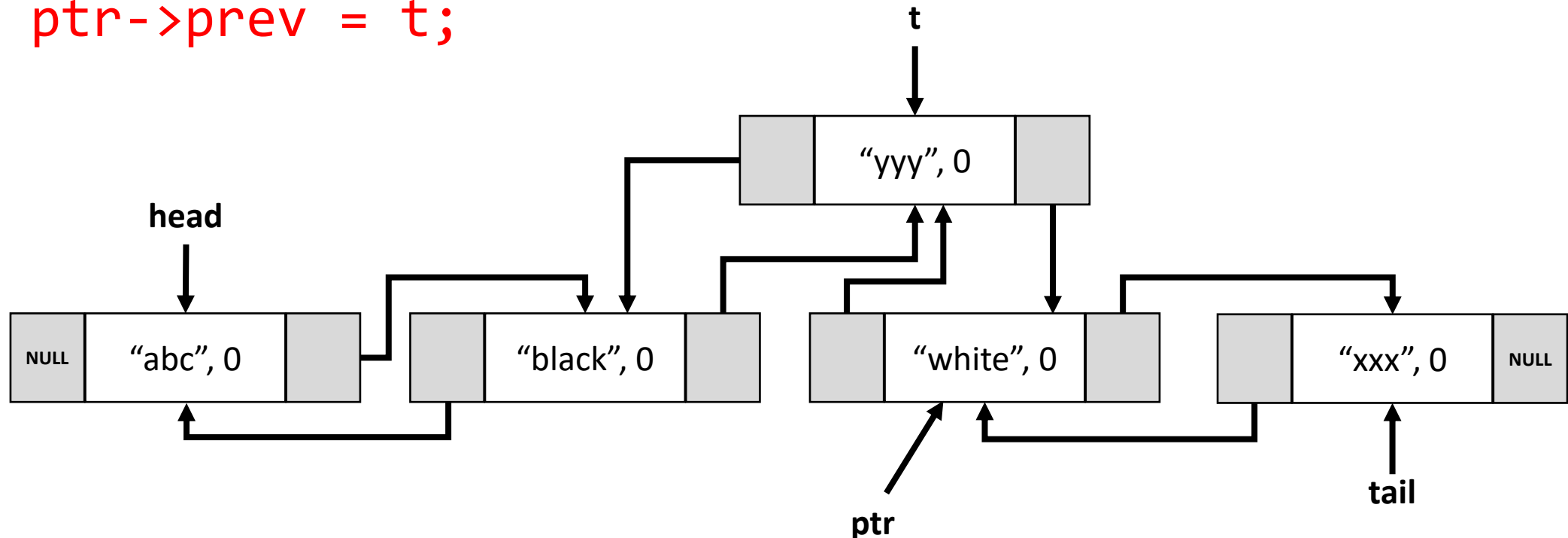
insertbefore (4/5)

- Η εισαγωγή στο ενδιάμεσο της λίστας πριν το δείκτη ptr.
 - Ένας νέος κόμβος πρόκειται να προστεθεί μεταξύ του ptr->prev και του ptr.
- Βήμα 3: αλλαγή του next για το ptr->prev:
ptr->prev->next = t;



insertbefore (5/5)

- Η εισαγωγή στο ενδιάμεσο της λίστας πριν το δείκτη ptr.
 - Ένας νέος κόμβος πρόκειται να προστεθεί μεταξύ του ptr->prev και του ptr.
- Βήμα 4: αλλαγή του ptr->prev:
ptr->prev = t;



Διαγραφή κόμβου (remove)

- removefront

- Δύο περιπτώσεις:

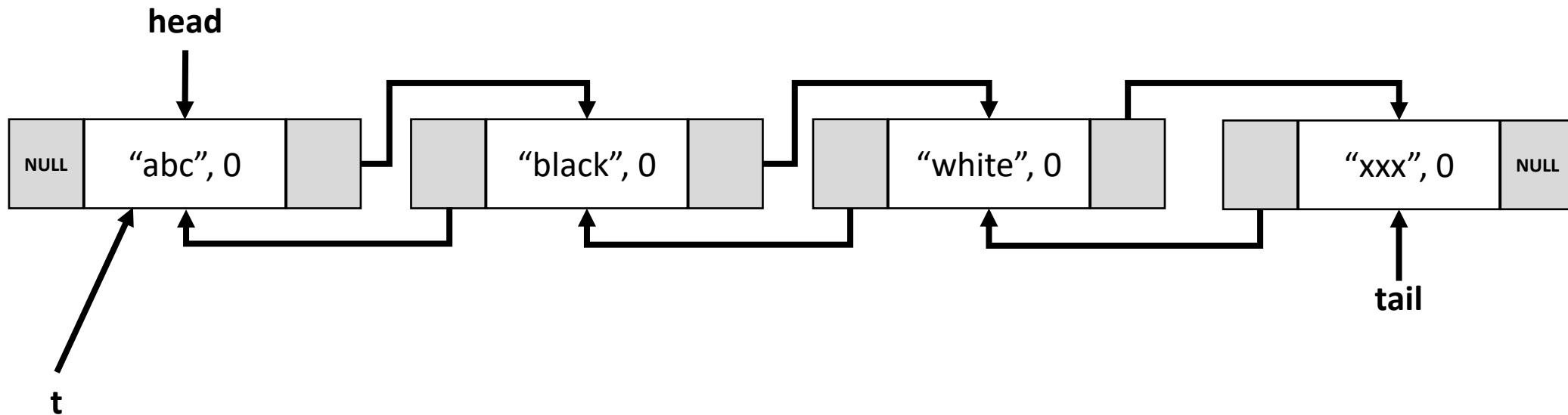
1. Η λίστα έχει ένα μόνο στοιχείο και συνεπώς μετά τη διαγραφή πρέπει να προκύψει η κενή λίστα.

```
if (head == tail) { // one item
    delete head;
    head = tail = NULL;
    size = 0;
    return;
}
```

2. Η λίστα έχει περισσότερα από ένα στοιχεία.

removefront (1/4)

- removefront
 - Η λίστα έχει περισσότερα από ένα στοιχεία
 - Βήμα 1: `listnode* t = head;`

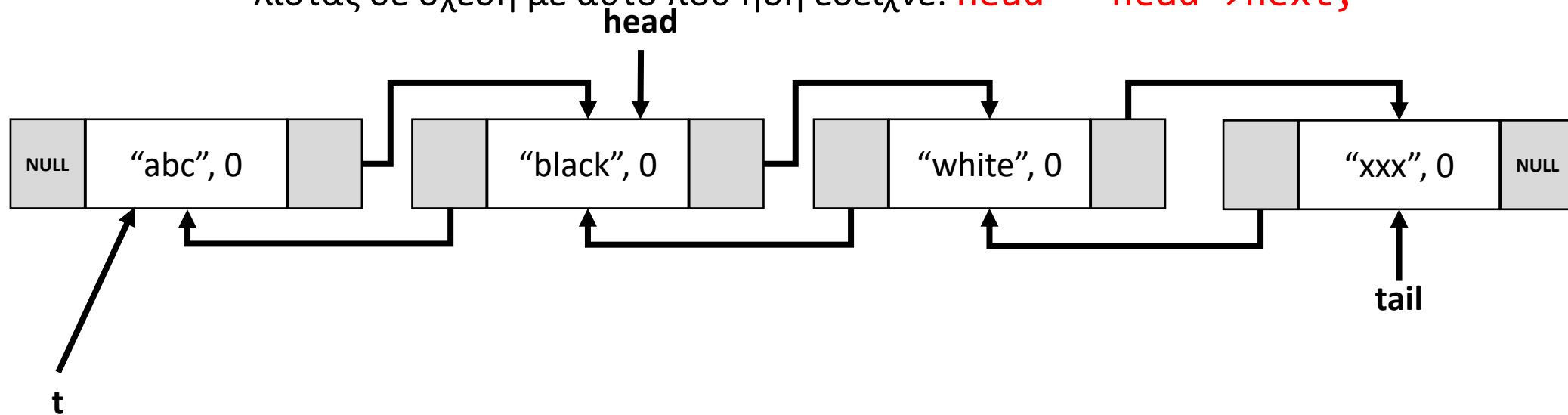


removefront (2/4)

- removefront

- Η λίστα έχει περισσότερα από ένα στοιχεία

- Βήμα 2: ο δείκτης head προχωρά έτσι ώστε να δείξει το αμέσως επόμενο στοιχείο της λίστας σε σχέση με αυτό που ήδη έδειχνε: `head = head->next;`

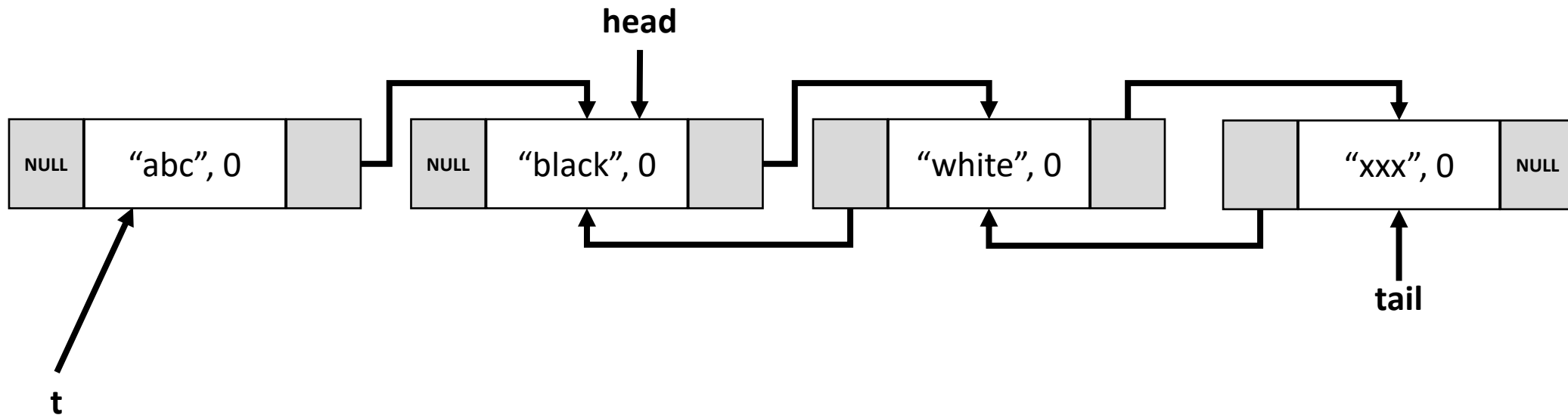


removefront (3/4)

- removefront

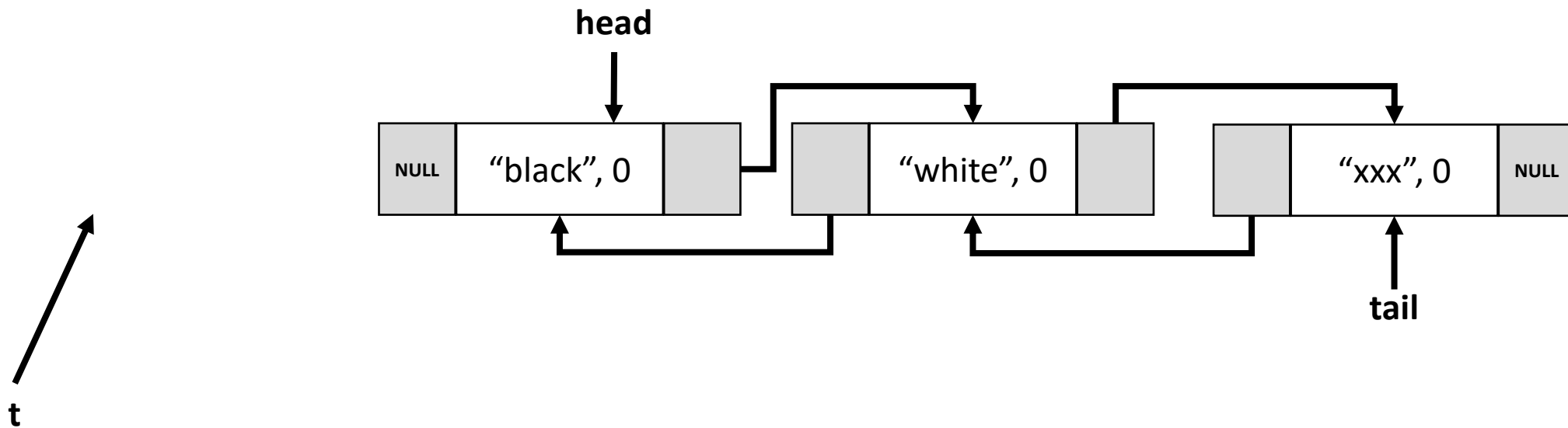
- Η λίστα έχει περισσότερα από ένα στοιχεία

- Βήμα 3: αποσύνδεση του δείκτη prev του head: `head->prev = NULL;`



removefront (4/4)

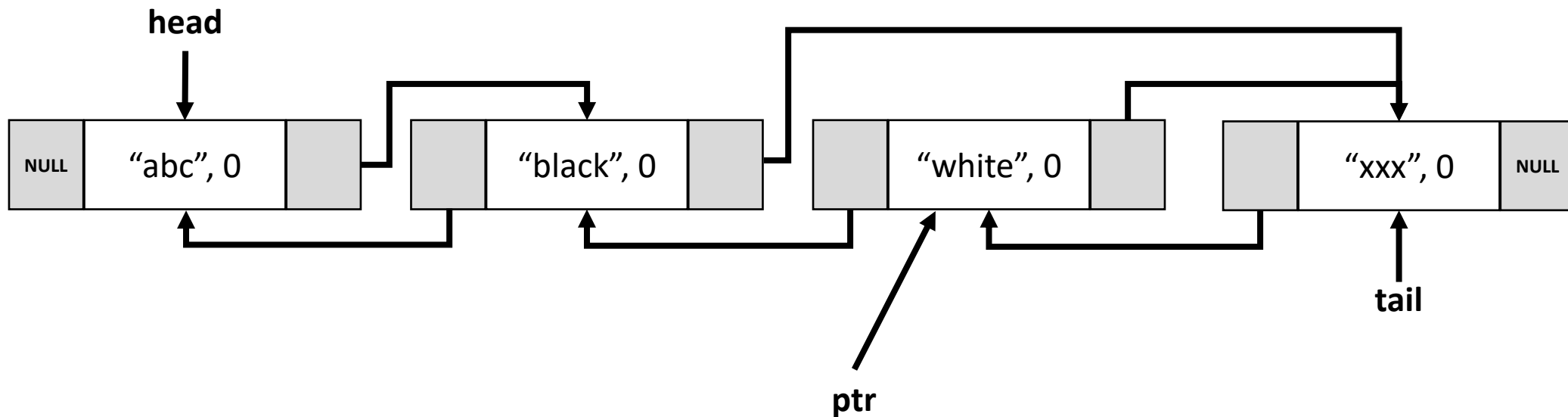
- removefront
 - Η λίστα έχει περισσότερα από ένα στοιχεία
 - Βήμα 4: **delete t;**



removemiddle (1/3)

- Διαγραφή του στοιχείου στο οποίο δείχνει ο δείκτης `ptr`.
 - Βήμα 1: αλλαγή του `next` για το δείκτη `ptr->prev`

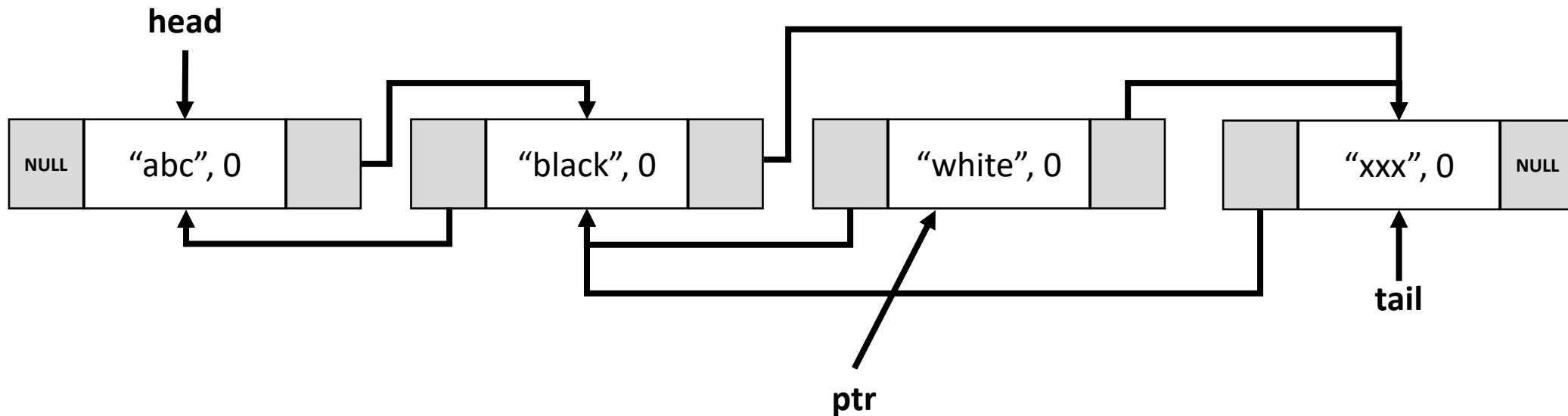
`ptr->prev->next = ptr->next;`



removemiddle (2/3)

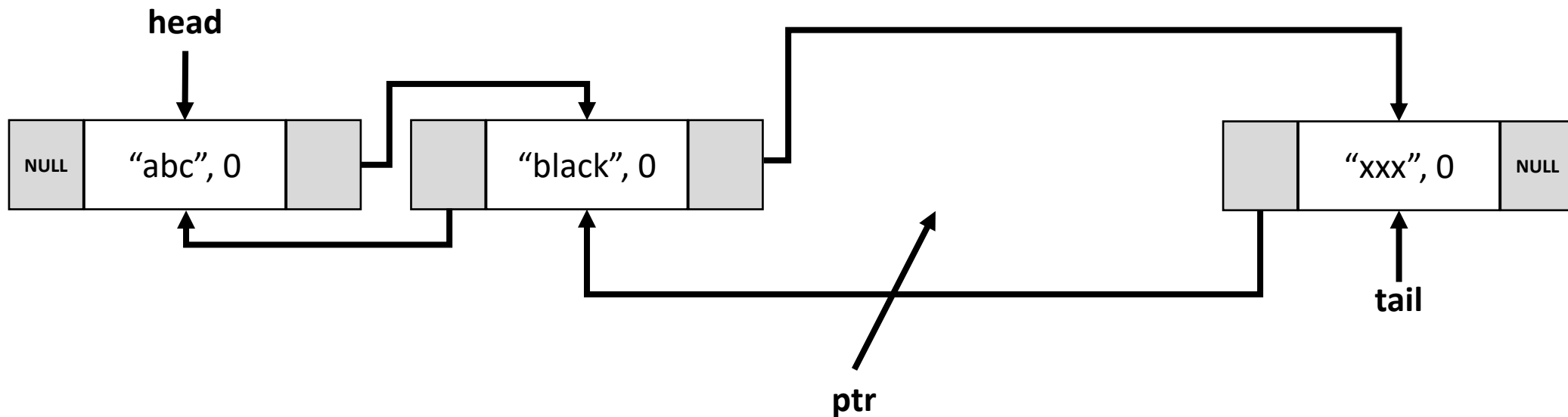
- Διαγραφή του στοιχείου στο οποίο δείχνει ο δείκτης `ptr`.
 - Βήμα 2: αλλαγή του `prev` για το δείκτη `ptr->prev`

`ptr->prev->prev = ptr->prev;`



removemiddle (3/3)

- Διαγραφή του στοιχείου στο οποίο δείχνει ο δείκτης ptr.
 - Βήμα 3: **delete ptr;**



Αναζήτηση (search)

- Χρησιμοποιώντας μια επανάληψη με `while` επιτυγχάνεται διάσχιση όλων των κόμβων της λίστας.

```
listnode *mylist::search(const string &s)
{
    listnode *t = head;

    while ((t!=NULL) && (t->s != s)) t = t->next;
    return t;
}
```

Ερωτήσεις σύνοψης

- Γιατί οι λειτουργίες εισαγωγής και διαγραφής έχουν αυξημένο υπολογιστικό κόστος στους στατικούς και στους δυναμικούς τύπους δεδομένων;
- Ποια είναι τα πλεονεκτήματα και ποια τα μειονεκτήματα των συνδεδεμένων λιστών;
- Τι πλεονεκτήματα και τι μειονεκτήματα έχουν οι διπλά συνδεδεμένες λίστες έναντι των απλά συνδεδεμένων λιστών;

Αναφορές

- <http://www.cs.fsu.edu/~xyuan/cop3330/>