C programming Language

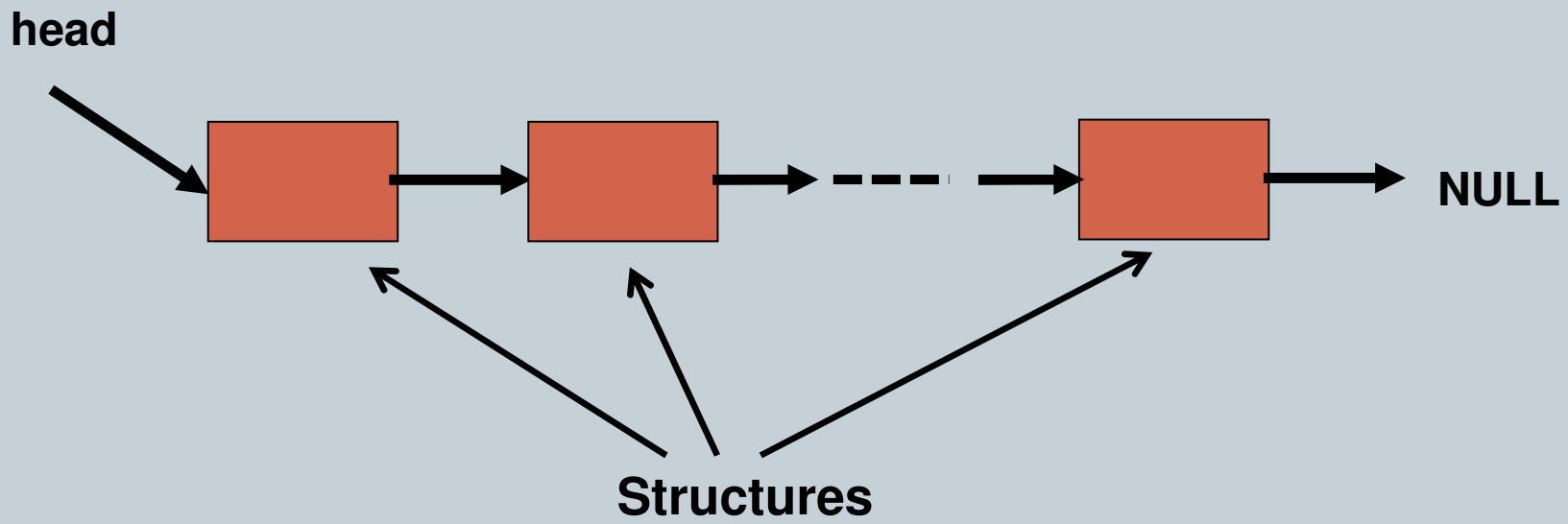
Chapter 4:

2. Linked Lists

Problems with dynamic arrays

- Problems:
  - Adding/ delete member.
  - Reallocation.
  - Building sort list .
  - Merging .
- Solution: Linked list
  - Simple add/delete member.
  - No need reallocation.
  - Building sorting.
  - Simple merging.

Linked lists ?



Linked list's member- example

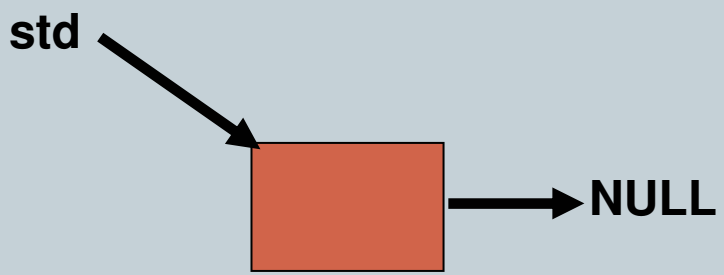
```
typedef struct item_t {
   char ID[ID_LENGTH];
   char Name[NAME_LENGTH];
   int grade;
// A pointer to the next item on the list
   struct Student_t *next;
} item;
```

Creating a new kind of student

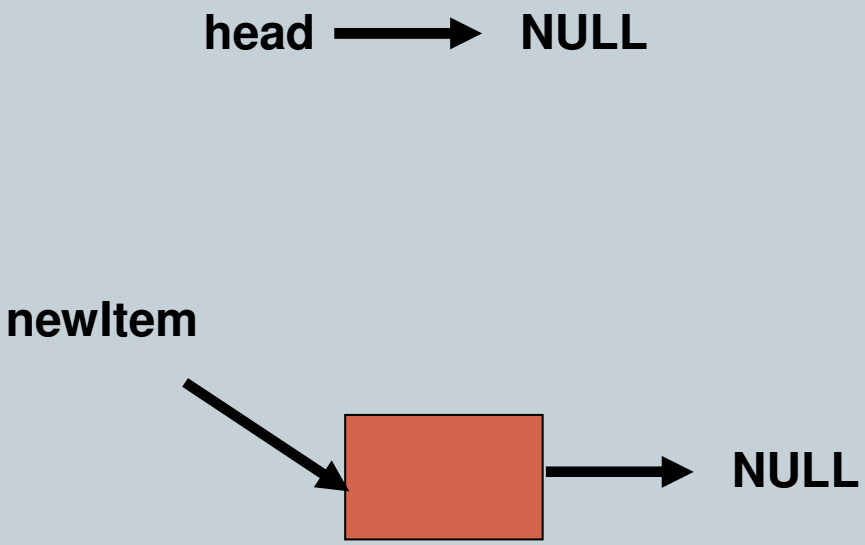
- Usually when using linked lists we don't know how many elements will be in the list
- Therefore we would like to be able to dynamically allocate new elements when the need arises.
- A possible implementation follows…

Creating a new kind of student

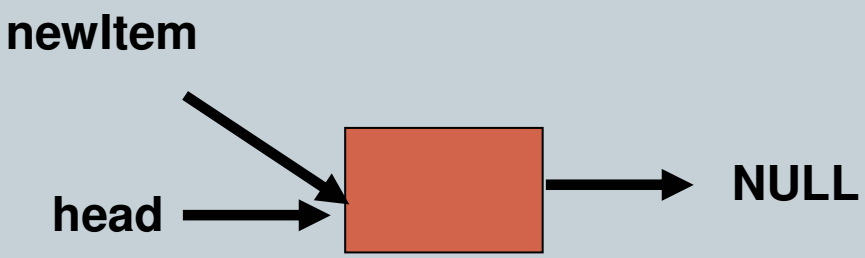
```
item*create_student(char * name, char  * ID, int grade) {
    item *std;
    std = (item *)malloc(item));
    if (std == NULL) {
            printf("Memory allocation error!\n");
            exit(1);
    }
    strcpy(std->Name, name);
    strcpy(std->ID, ID);
    std->grade = grade;
    std->next = NULL;
    return std;
}
```



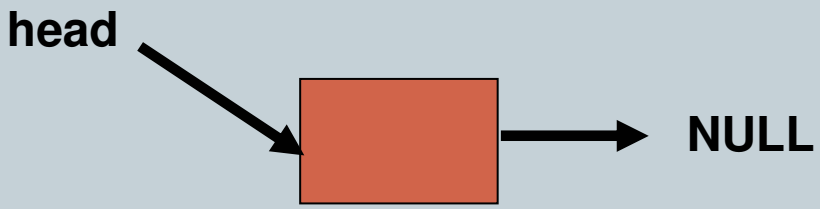
Adds a new item to the end of the list



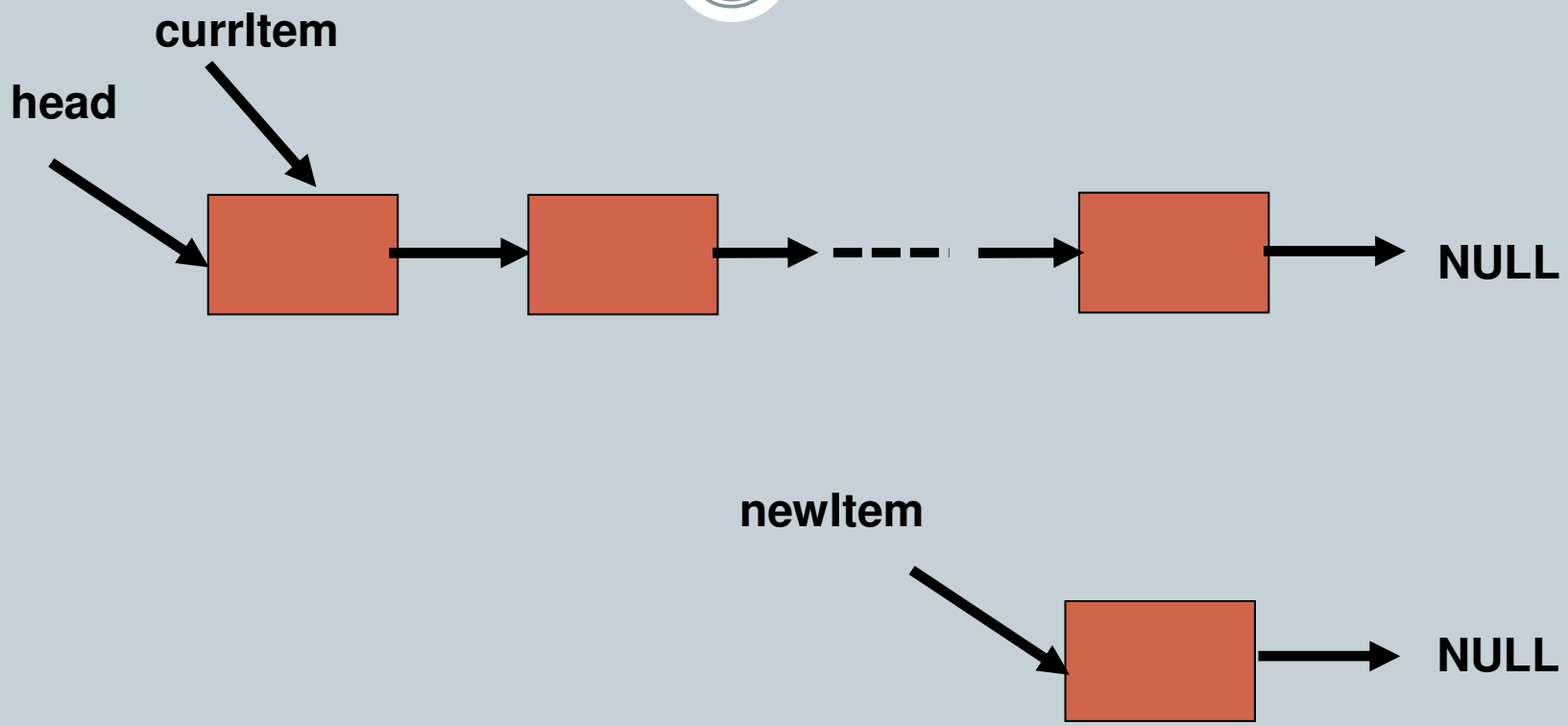
Adds a new item to the end of the list



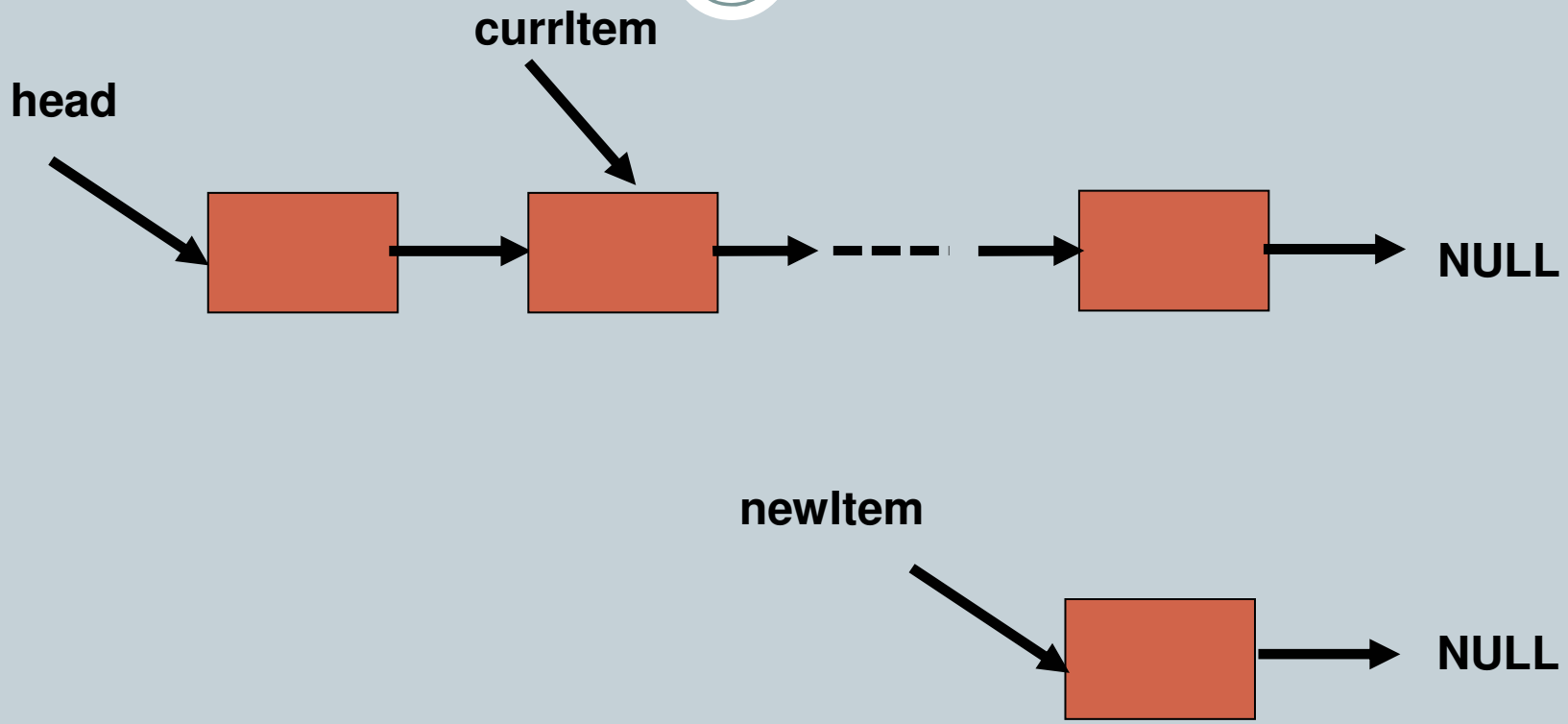
Adds a new item to the end of the list



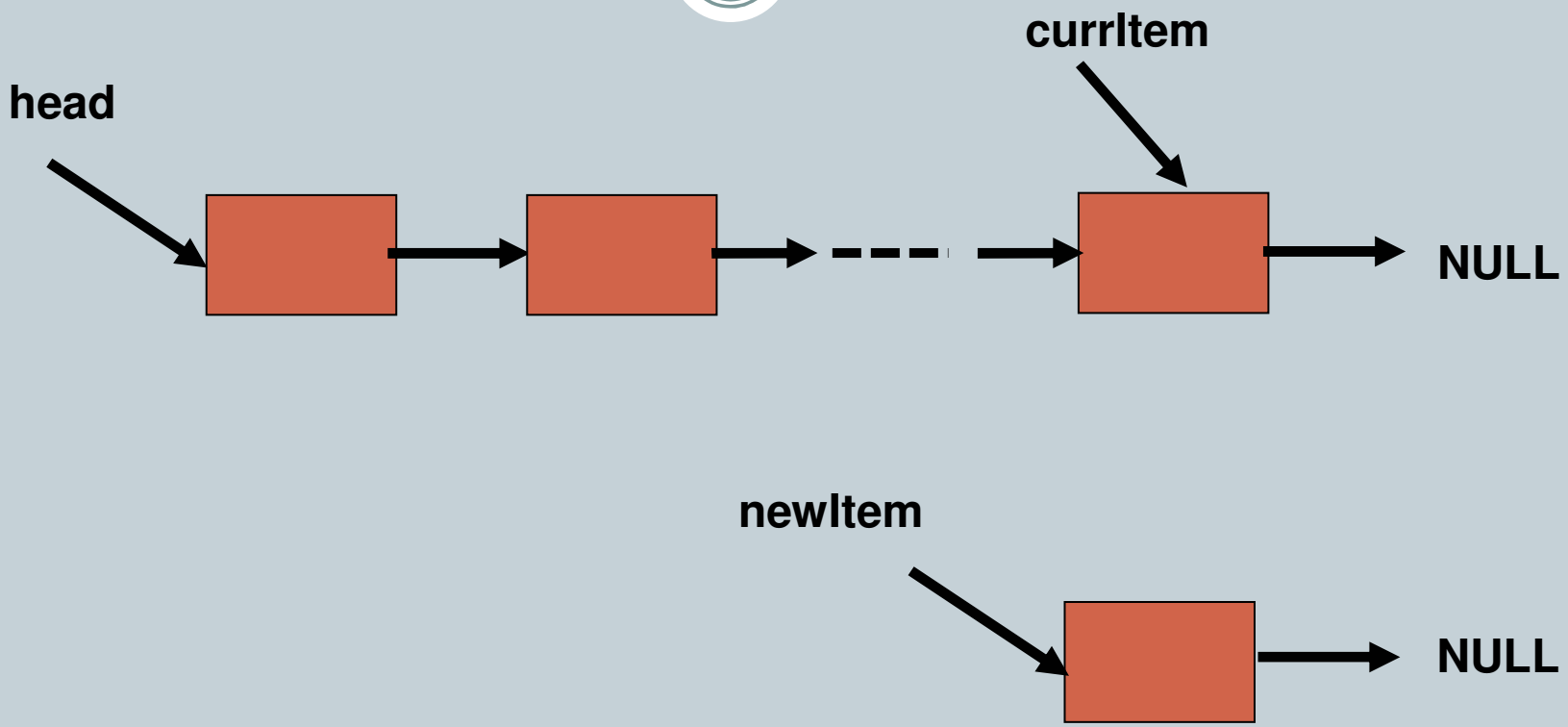
Adds a new item to the end of the list



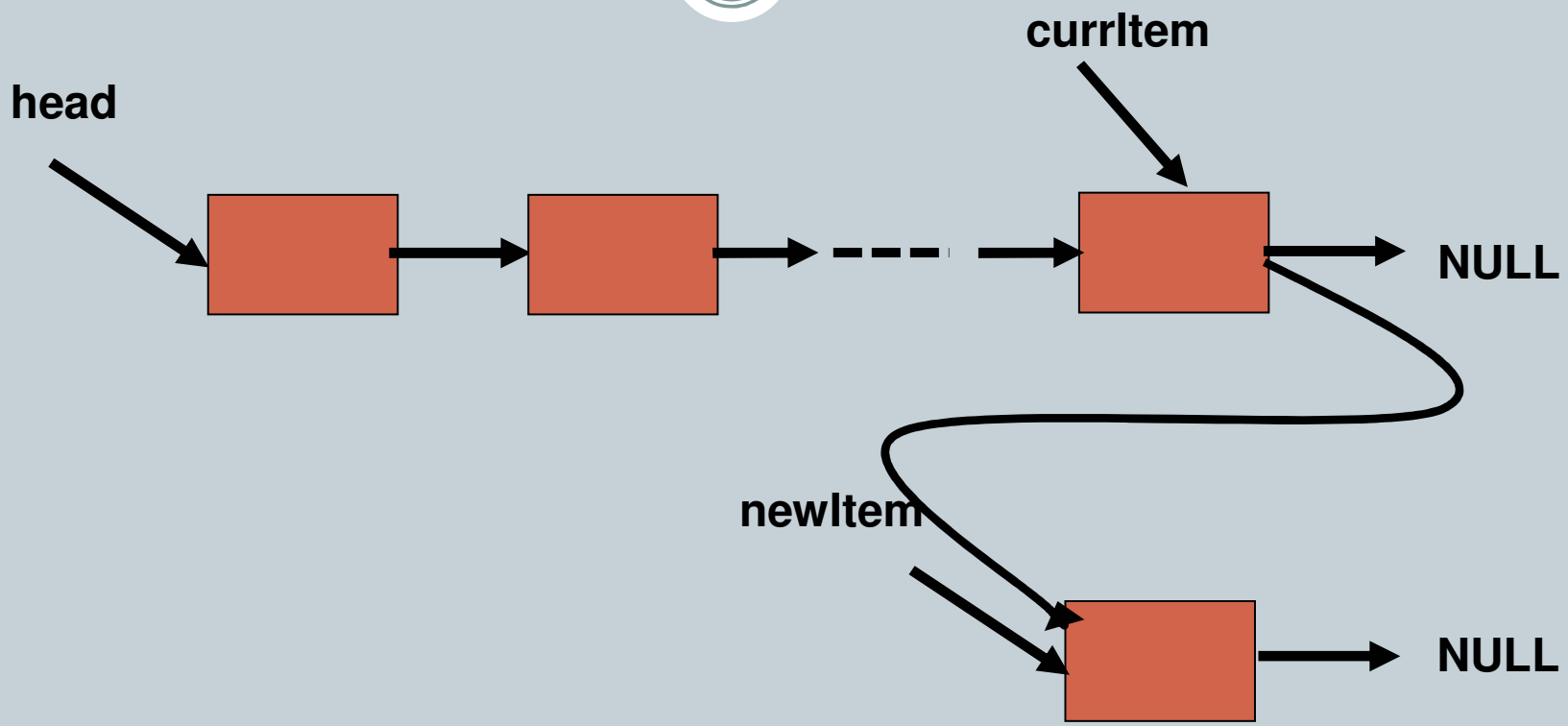
Adds a new item to the end of the list



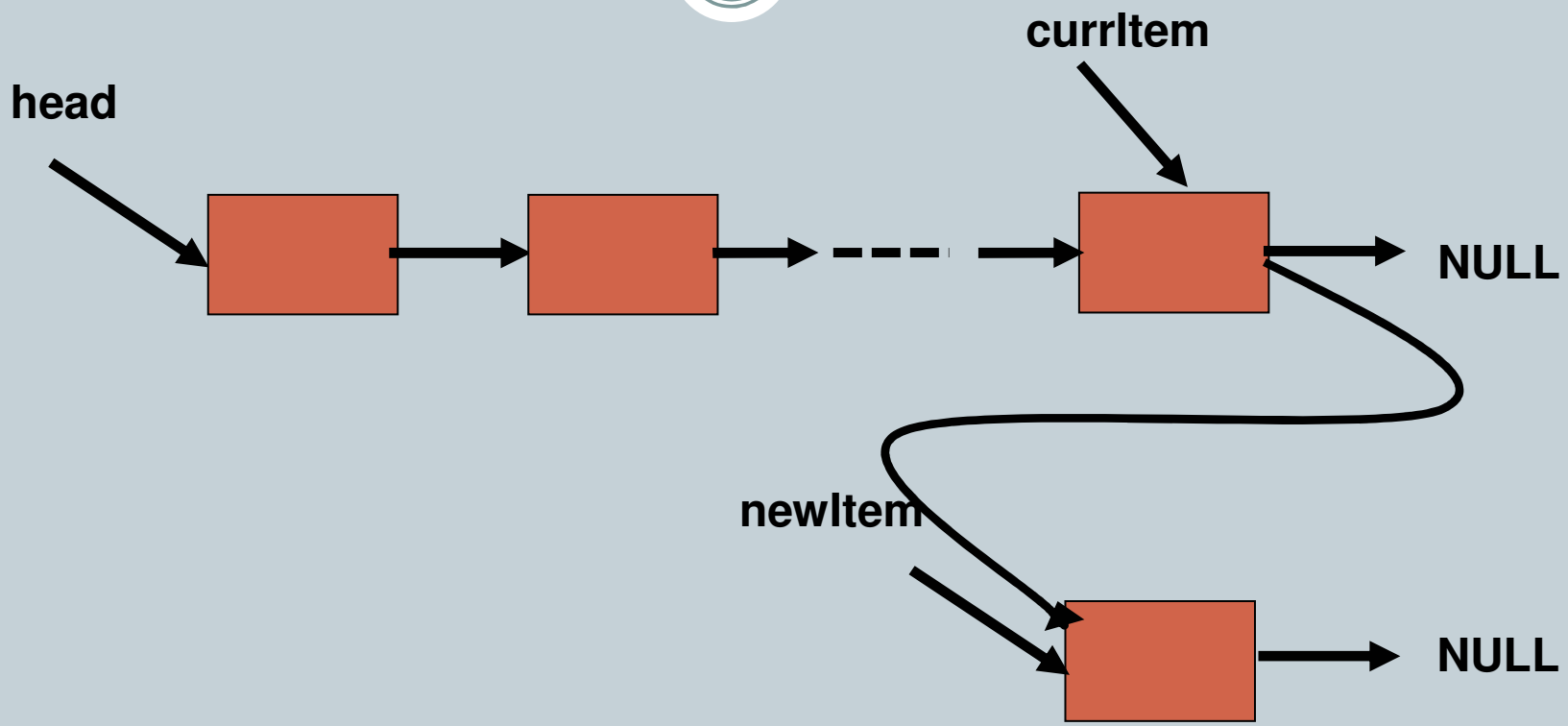
Adds a new item to the end of the list



Adds a new item to the end of the list



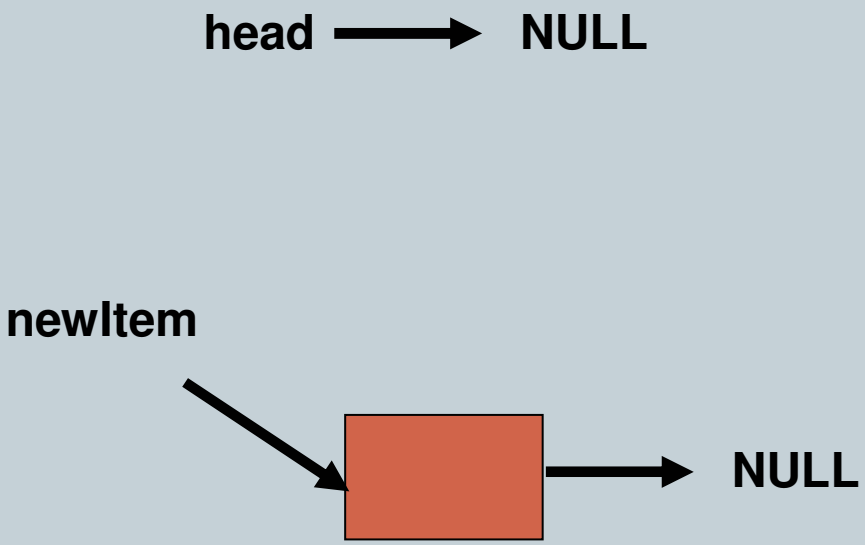
Adds a new item to the end of the list



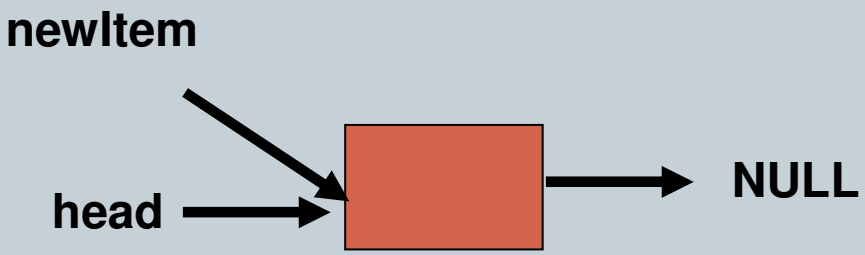
Adds a new item to the end of the list

```
item * add_last(item *head,  item* newItem){
        item *currItem;
        if (!head)
                return newItem;
        currItem = head;
        while(currItem->next)
                currItem = currItem->next;
        currItem->next = newItem;
   return head;
}
```

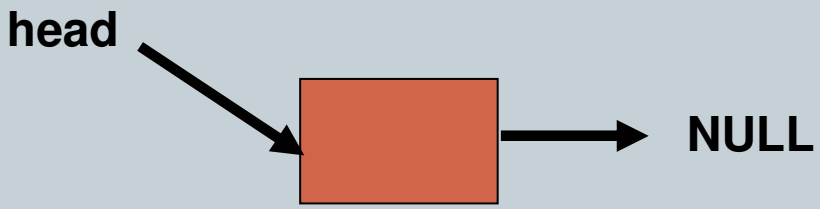
Inserts into a sorted list



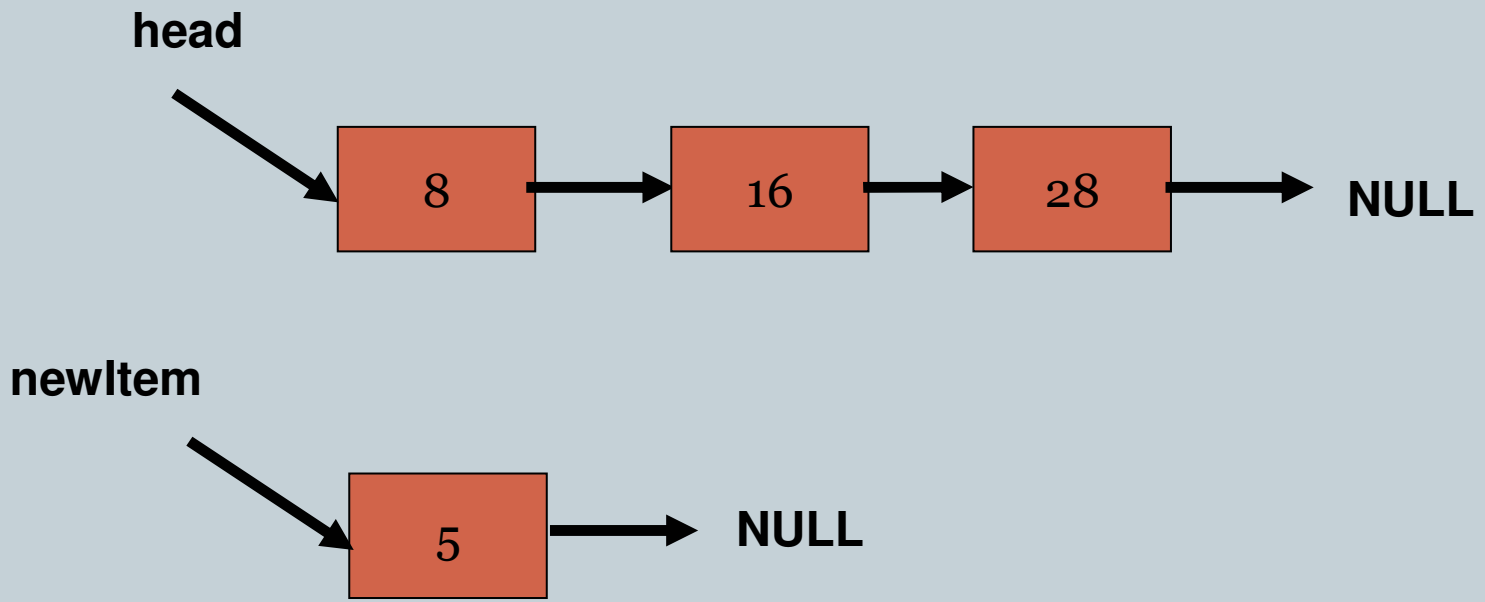
Inserts into a sorted list



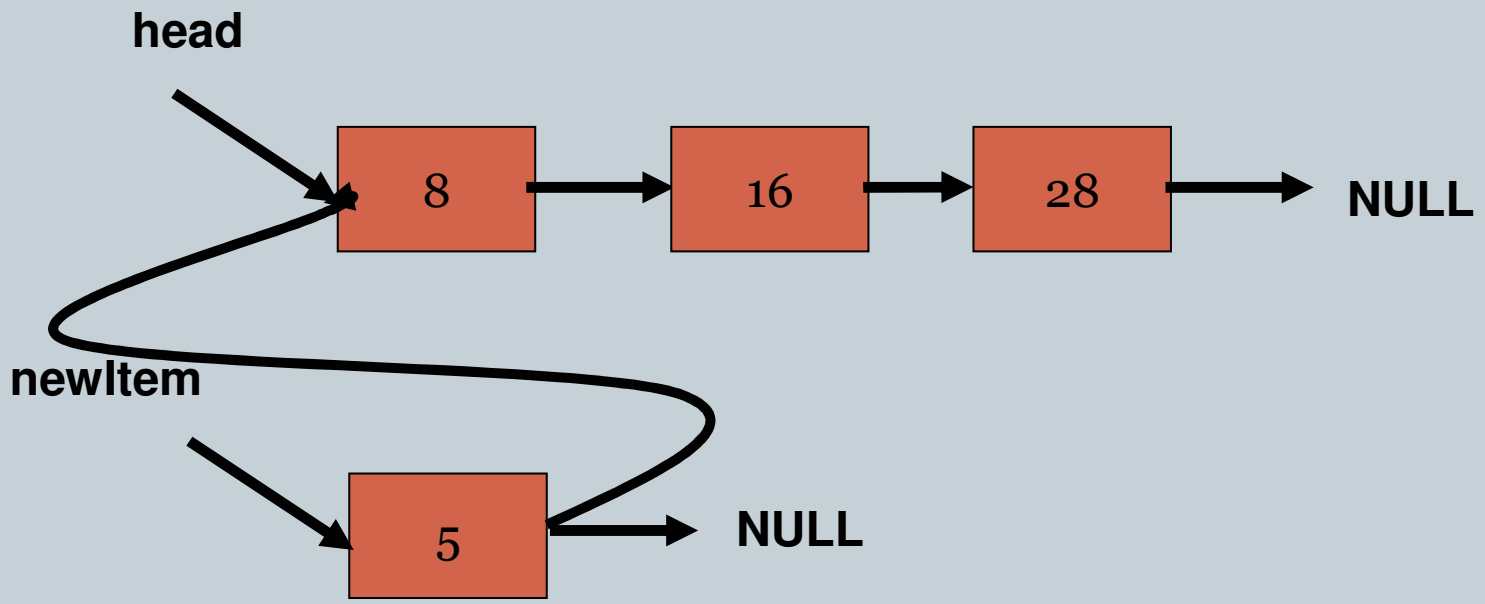
Inserts into a sorted list



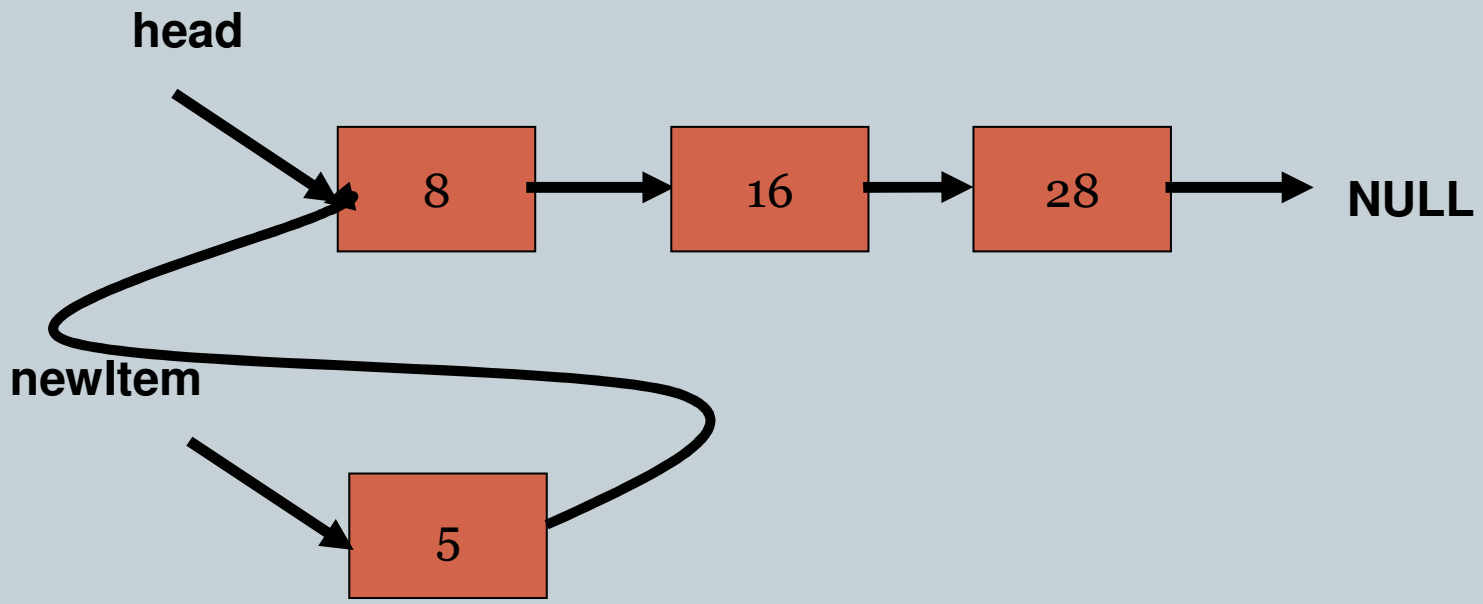
Inserts into a sorted list



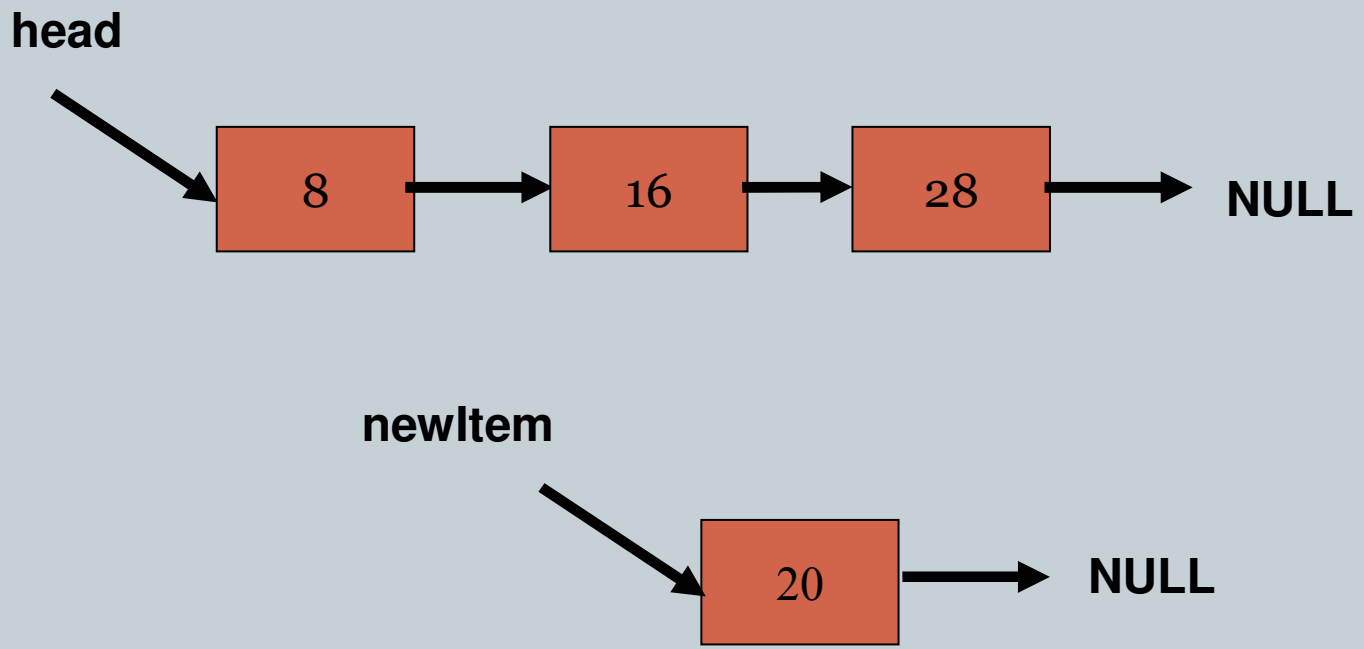
Inserts into a sorted list



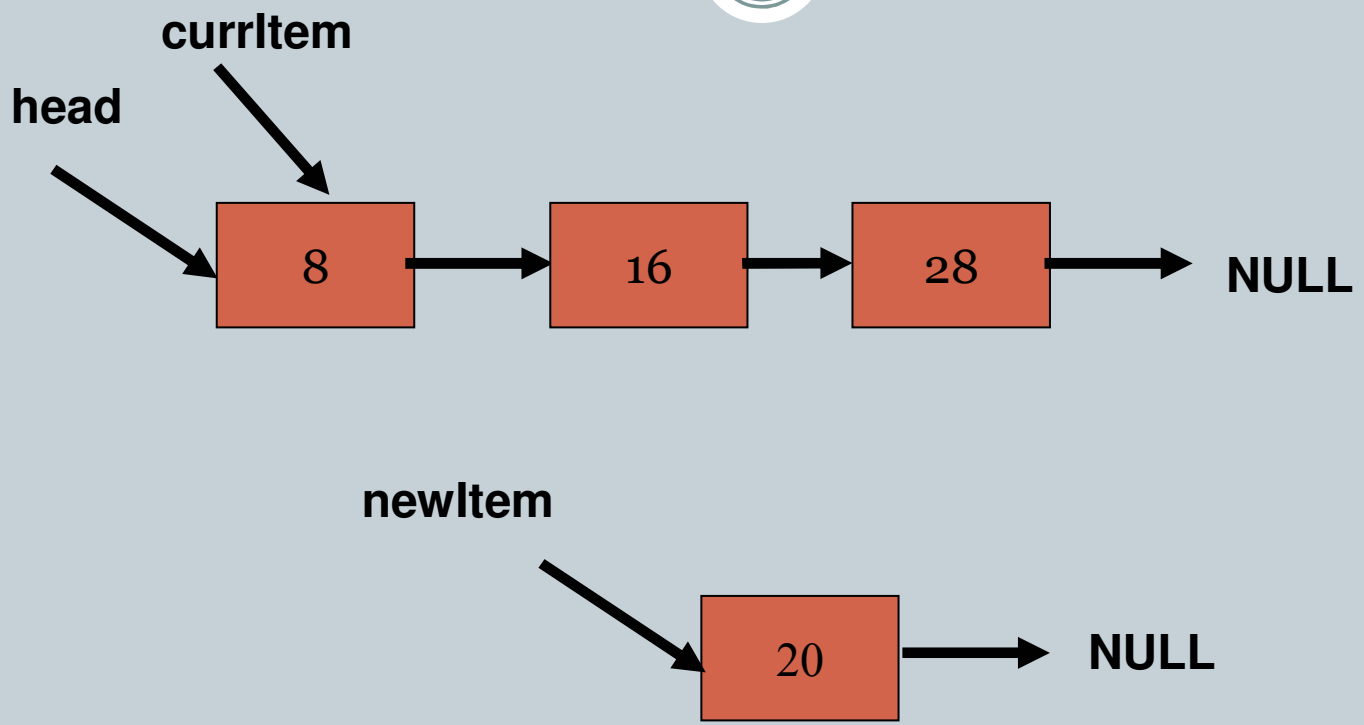
Inserts into a sorted list



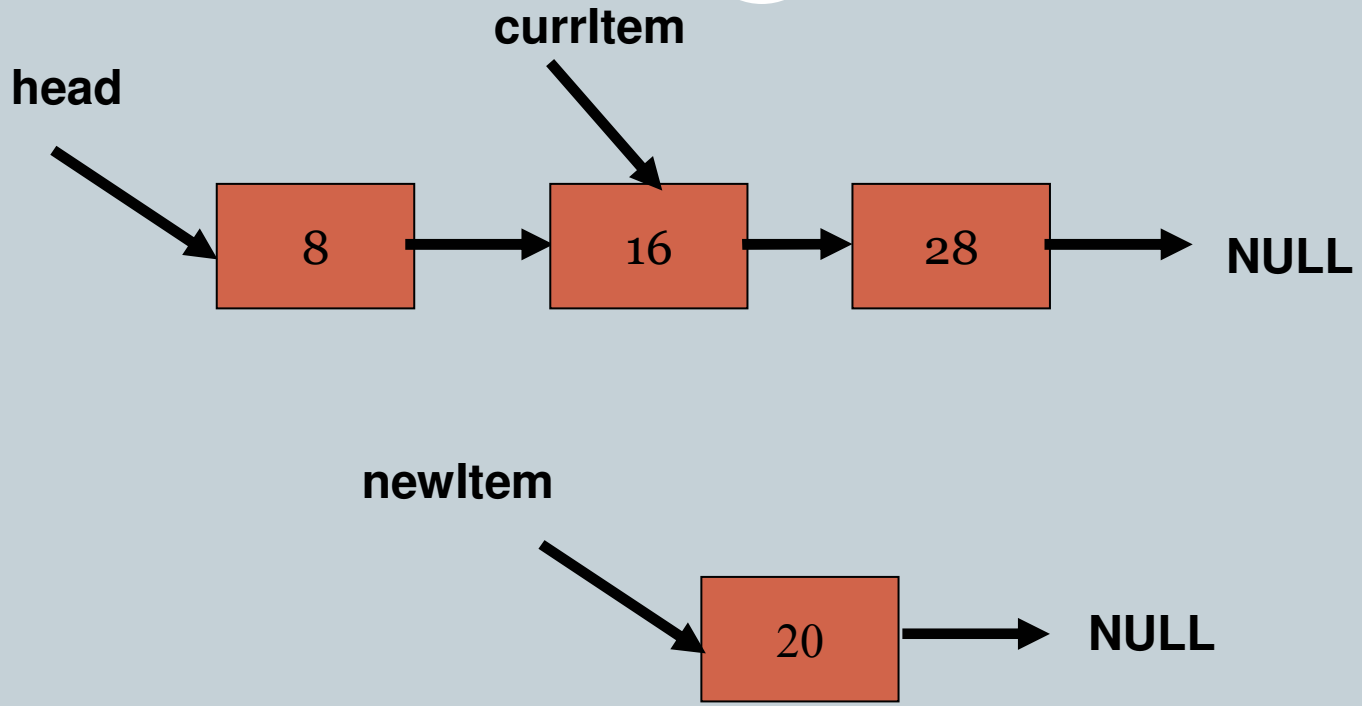
Inserts into a sorted list



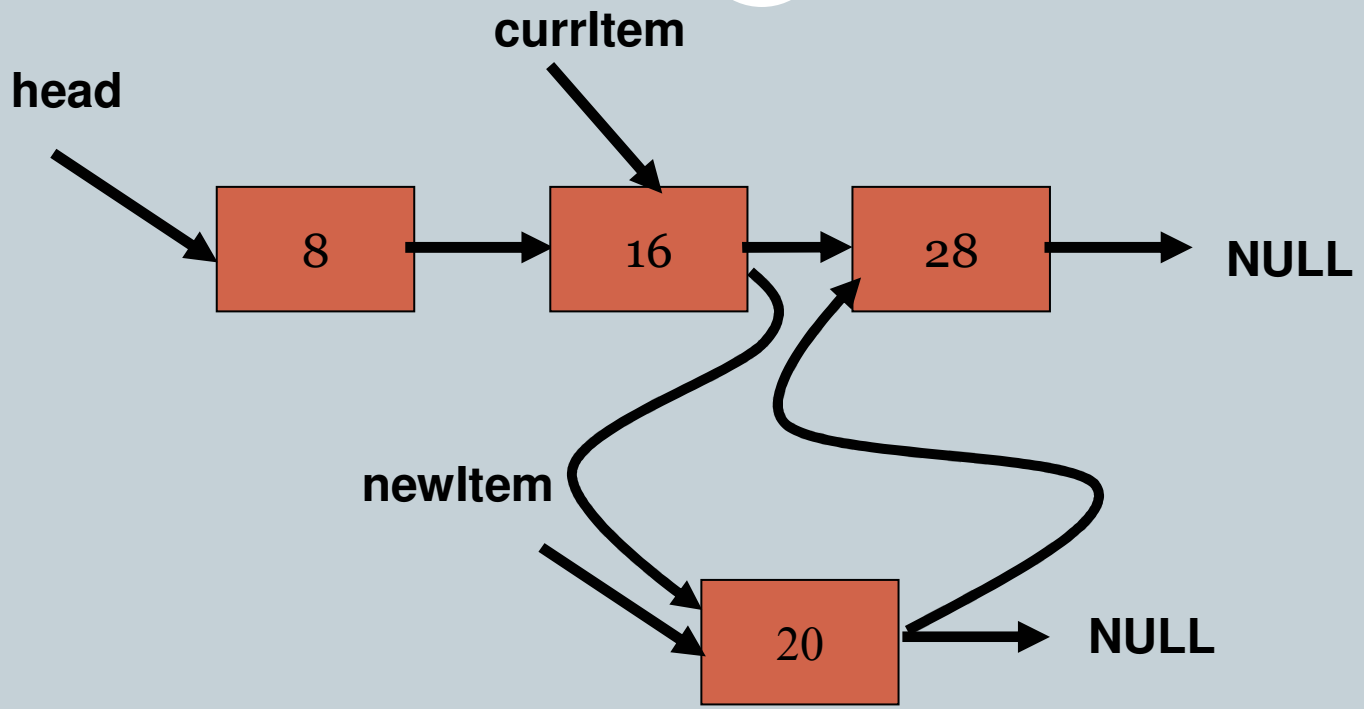
Inserts into a sorted list



Inserts into a sorted list



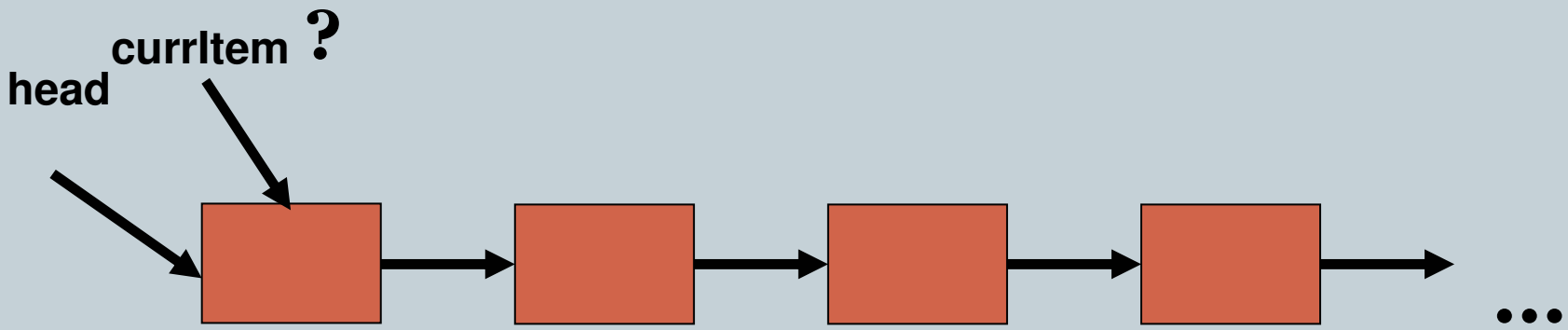
Inserts into a sorted list



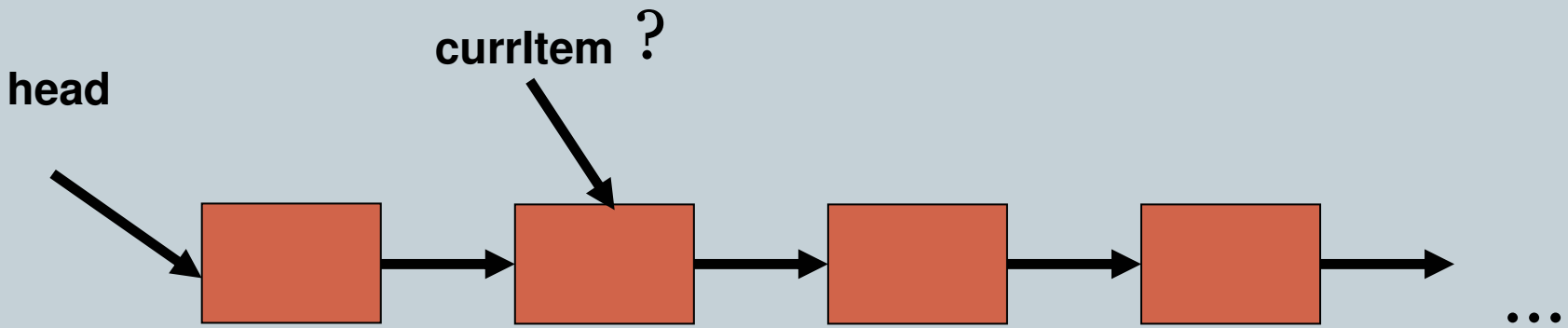
Inserts into a sorted list

```
// while keeping it sorted ascending by key
item * insert(item *head, item *newNode) {
  item *currItem;
  if (!head)
     return newNode;
//check if newNode's key is smaller than all keys and should be first
   if (newNode->key < head->key)   {
       newNode->next = head;
       return newNode;
    }
    currItem = head;
    while (currItem->next && newNode->key > currItem->next->key)
            currItem = currItem->next;
//put newNode between currItem and currItem->next
//(if currItem is last then currItem->next == NULL)
   newNode->next = currItem->next;
   currItem->next = newNode;
    return head;
}
```

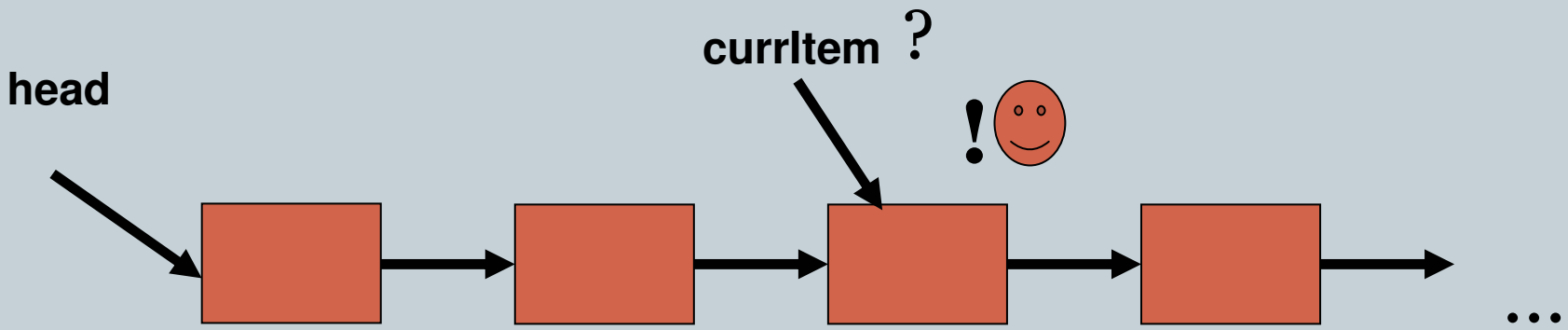
Linked lists - searching



Linked lists - searching



Linked lists - searching



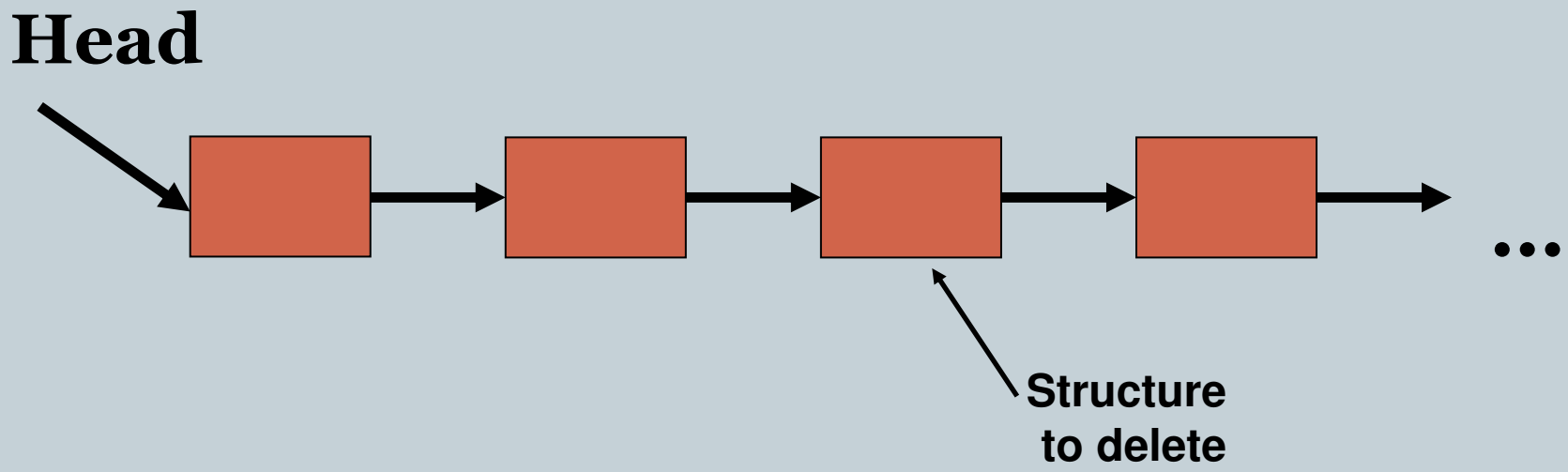
Searching for an item

```
 //searches for an item with passed key.
//returns NULL if didn't find it.
item *search(item *head, int key) {
        item *currItem = head;
        if (!head) return NULL;
        while (currItem) {              //loop through the list
                if (currItem->key == key)
                        return currItem;
                currItem = currItem->next;
        }       //didn't find the item with the requested key
        return NULL;
}
```

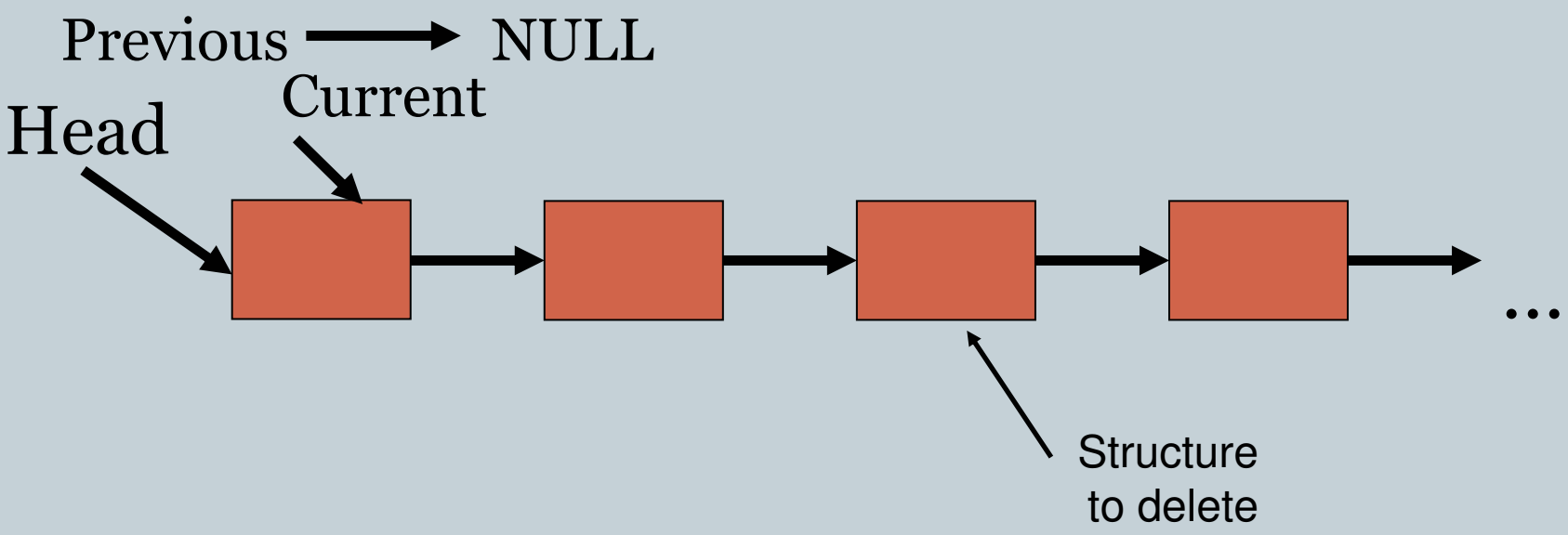
Print list's members

```
//prints keys of items of the list, key after key.
void printKeys(item *head) {
   item *curr = head;
   while (curr)   {
         printf("%d ", curr->key);
         curr = curr->next;
   }
   putchar('\n');
}
```

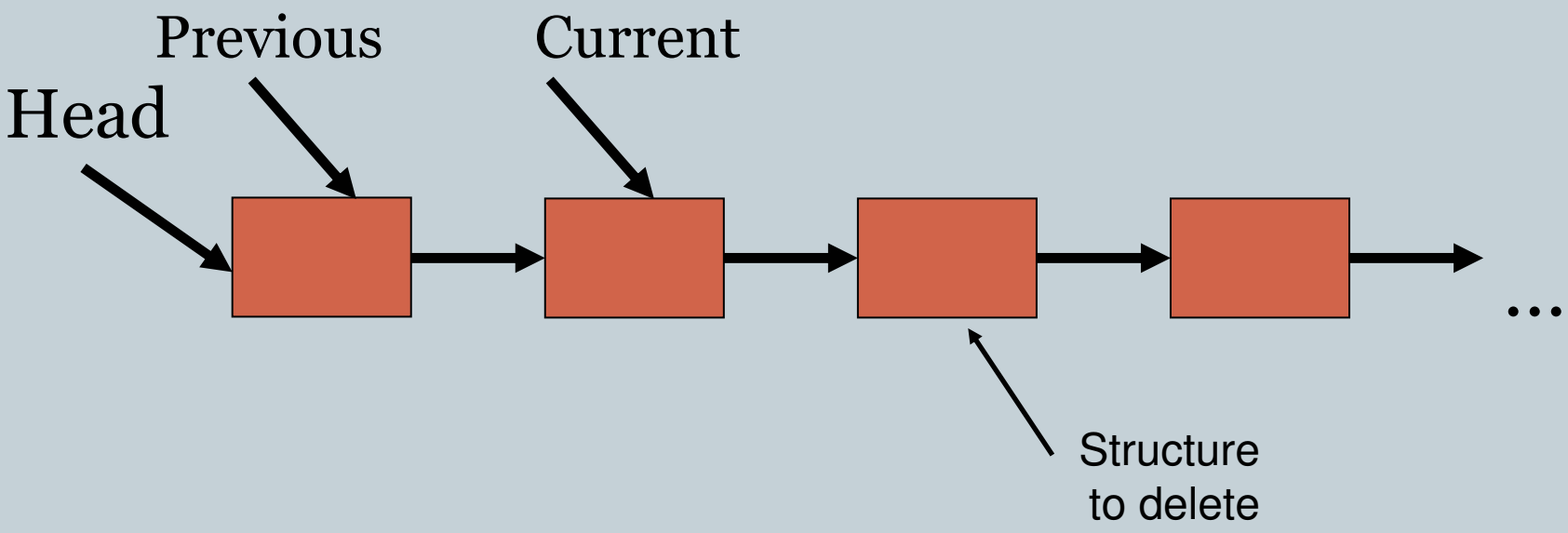
Linked lists - delete



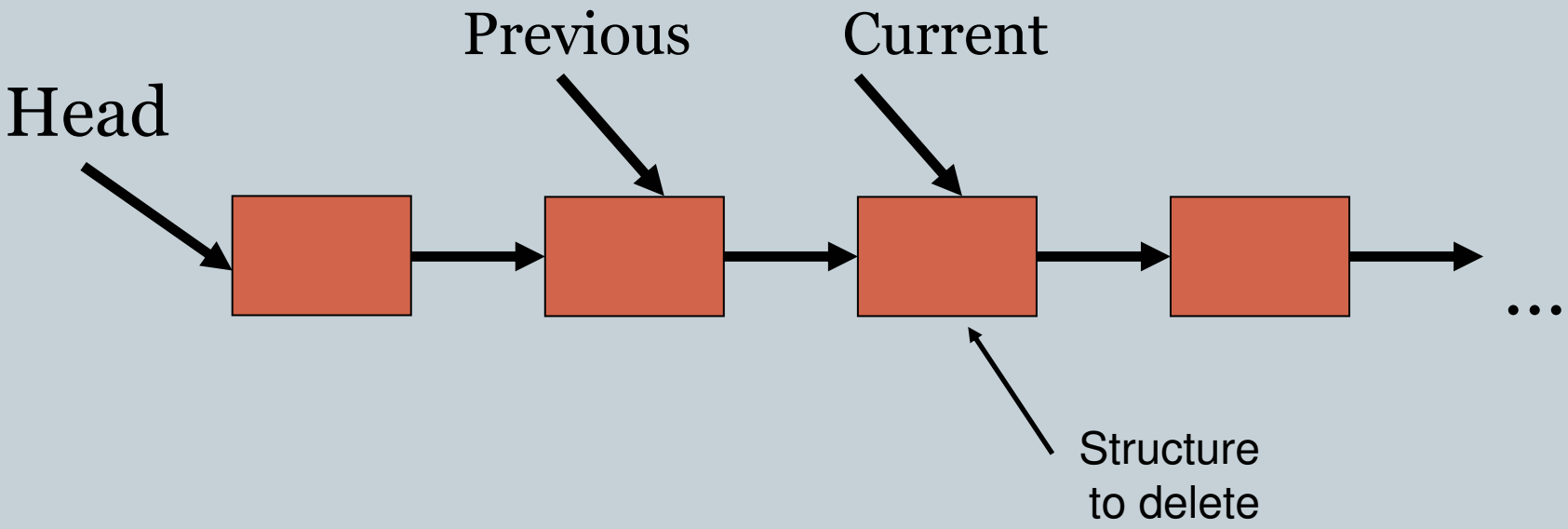
Linked lists - delete



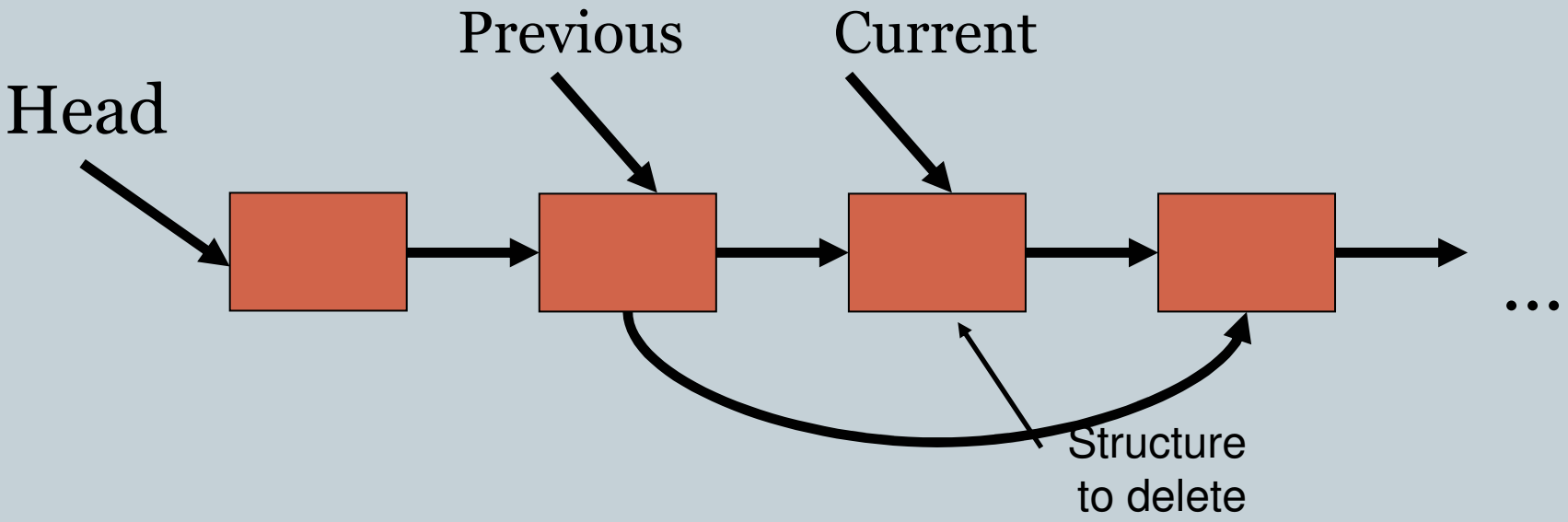
Linked lists - delete



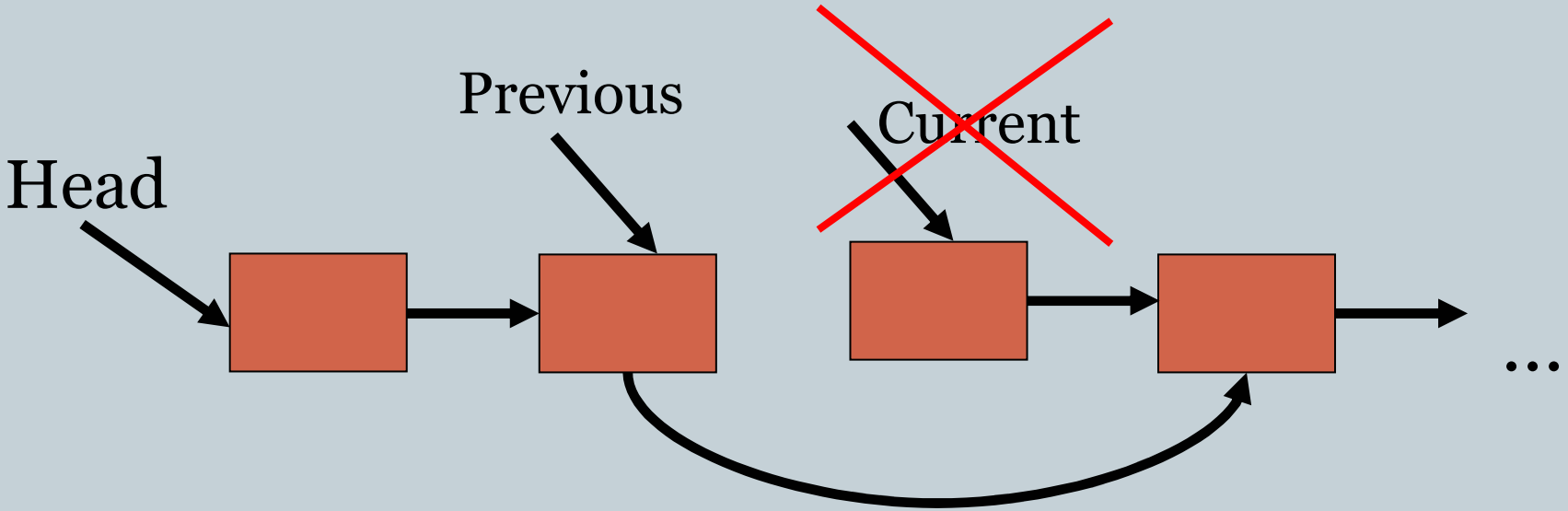
Linked lists - delete



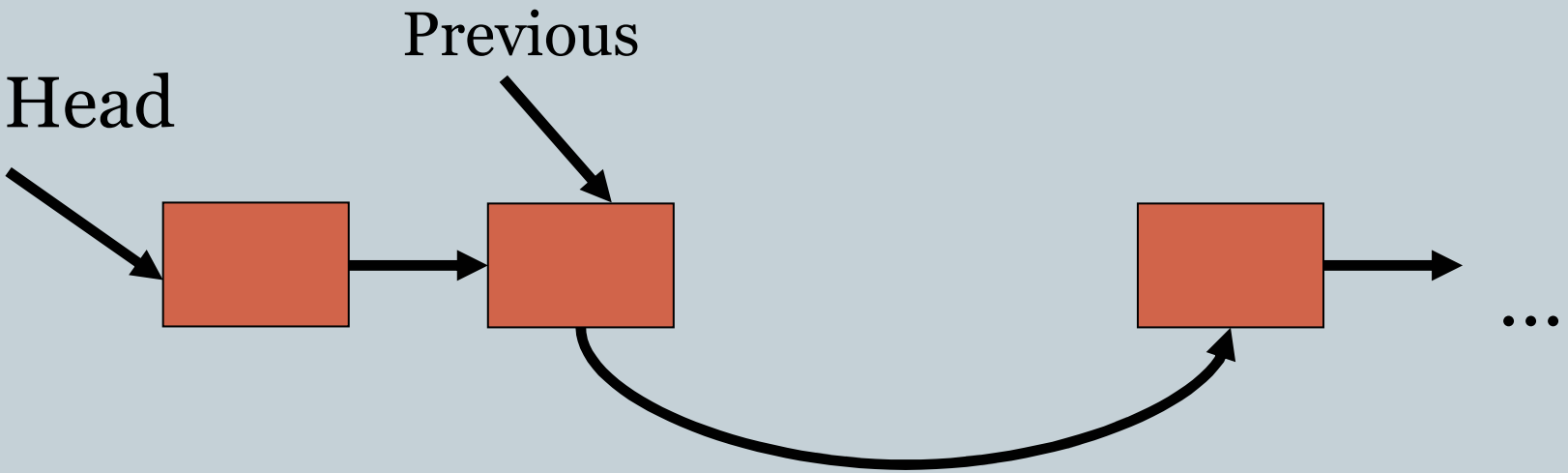
Linked lists - delete



Linked lists - delete



Linked lists - delete



Remove the item with a given value

```
item *remove(int value, item* head){
    item * curr= head,*prev=NULL;
    int found=0;
    if(!head)
            printf("The LL is empty\n");
    else{
            while(curr)
                    if(value==curr->value){
                            prev ?prev->next=curr->next:head=head->next;
                            free(curr);
                            found=1;
                            break;
                    }
                    else{
                            prev=curr;
                            curr=curr->next;
                    }
            if(!found)              printf("The record with key %d was not found\n",value);
    }
            return head;
}
```

Delete list

```
  //deletes and frees all items in list
void emptyList(item *head) {
   item *temp, *curr = head;
   while (curr)   {
         temp = curr;
         curr = curr->next;
         free(temp);
   }
}
```

Recursive freeing

- A perhaps simpler way to free a list is recursively.

```
void free_list(item *head)  {
   if (head== NULL)      // Finished freeing. Empty list
          return;

   free_list(head->next);   // Recursively free what's ahead
   free(head);
}
```

Merging sorted lists

```
NODE * merge( NODE* list1, NODE* list2) {
   if( list1 == NULL)
         return( list2);
   if( list2 == NULL) return( list1);
   if( list1->val <= list2->val) {
         list1->next = merge( list1->next, list2);
         return( list1);
   }
   else {
         list2->next = merge( list1, list2->next);
         return( list2);
   }
}
```

Chapter 4:

3. Binary trees

Tree Structure

- **Tree structure is a way of representing the hierarchical nature of a structure in a graphical form.**
- **Mathematically, a tree is an acyclic connected graph where each node has zero or more *children* nodes and at most one *parent* node. Furthermore, the children of each node have a specific order.**
- **In computer science, a tree is a widely-used data structure that emulates a hierarchical tree structure with a set of linked nodes.**

Binary trees : definitions

- **Binary tree is a tree data structure in which each node has at most two child nodes, usually distinguished as "left" and "right".**
- **Nodes with children are parent nodes, and child nodes may contain references to their parents.**
- **A node that has a child is called the child's parent node (or *ancestor node*, or superior). A node has at most one parent.**
- **Nodes that do not have any children are called leaf nodes.**

Binary trees : definitions (cont.)

- **The height of a node is the length of the longest downward path to a leaf from that node.**
- **The height of the root is the height of the tree.**
- **The depth of a node is the length of the path to its root (i.e., its *root path*).**
- **An internal node or inner node is any node of a tree that has child nodes and is thus not a leaf node.**
- **A subtree of a tree *T* is a tree consisting of a node in *T* and all of its descendants in *T*.**

Common operations

- Enumerating all the items .
- Enumerating a section of a tree.
- Searching for an item.
- Adding a new item at a certain position on the tree.
- Deleting an item.
- Removing a whole section of a tree (called pruning) .
- Adding a whole section to a tree (called grafting) .
- Finding the root for any node .

Node – Definition and making

- typedef struct node_t {
  int data;
  struct node_t *right, *left;
  }node;

- node *make(int y) {
  node *newnode;
  newnode=(struct node *)malloc(sizeof(struct node));
  newnode->data=y;
  newnode->right=newnode->left=NULL;
  return(newnode);
  }

Making Tree

```
void left(node *r, int x){
   if(r->left  != NULL)
        printf("\n Invalid !");
   else
        r->left=make(x);
}
```

```
void right( node *r, int x){
     if(r->right  != NULL)
          printf("\n Invalid ");
     else
          r->right=make(x);
}
```

Making Sort Tree

```
void main(){
    node *root,*p,*q;
    int no;
    int choice;
    printf("\n Enter the root:");
    scanf("%d",& no);
    root=make(no);
    p=root;
    while(1){
      printf("\n Enter  number:");
      scanf("%d",& no);
      if(no==-1)
          break;
      p=root;
      q=root;
```

```
      while(no!=p->data && q!=NULL){
            p=q;
            if(no<p->data)
                  q=p->left;
            else
                  q=p->right;
       } // end while
       if(no<p->data){
            printf("\n Left branch of %d
                  is %d",p->data,no);
            left(p,no);
       }
       else  {
            right(p,no);
            printf("\n Right Branch of
                  %d is %d",p->data,no);
       }
    }// end while (1)
} // end main
```

- inorder – בן שמאלי, אב , בן ימני.
- preorder – אב, בן שמאלי ,בן ימני.
- postorder – בן שמאלי, בן ימני, אב.

Inorder

```
void inorder(node *r){
        if(r!=NULL){
                inorder(r->left);
                printf("\t %d",r->data);
                inorder(r->right);
        }
}
```

Preorder

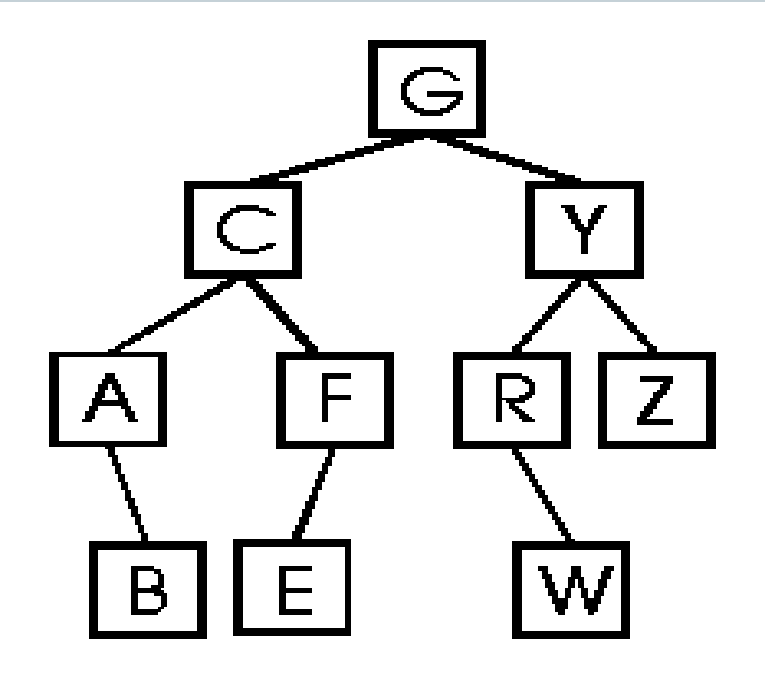
```
void preorder( node *r)  {
        if(r!=NULL) {
                printf("\t %d",r->data);
                preorder(r->left);
                preorder(r->right);
        }
 }
```

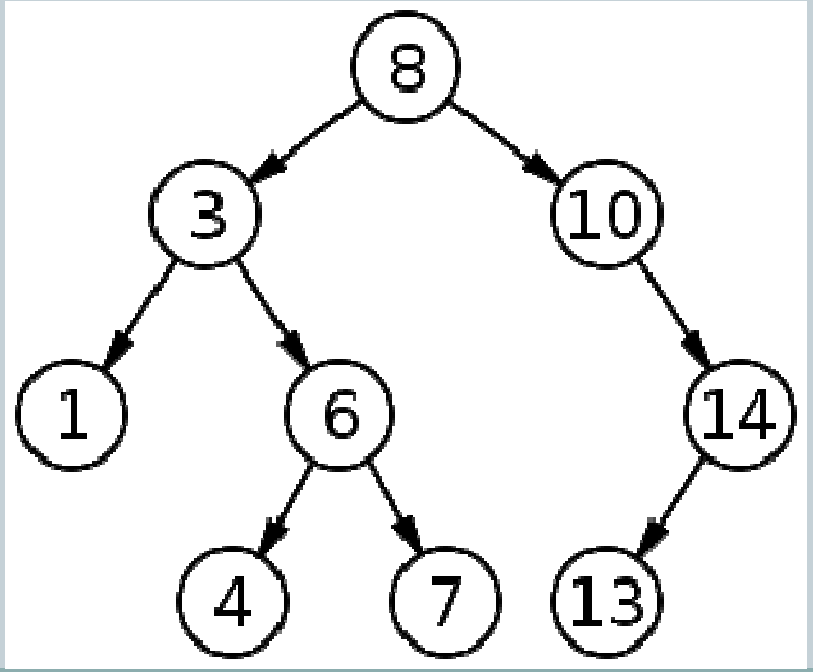
Postorder

```
void postorder( node *r)  {
        if(r!=NULL)  {
                postorder(r->left);
                postorder(r->right);
                printf("\t %d",r->data);
        }
}
```

Binary search tree – עץ חיפוש בינארי

- עץ בינארי אשר בו עבור כל צומת, הערכים של כל האיברים בתת העץ השמאלי שלו קטנים (או שווים) ממנו, וכל האיברים בתת העץ הימני שלו גדולים ממנו.





Insert new node in a binary search tree

```
void insert_info_by_key(TreeNode ** root, Data data){
    if (!(*root)){
        *root = create_tree_node(data);     //get new node for tree
    }
    else   {
       if (data.key == (*root)->TreeNode_info.key) return; //not duplicates
      if (data.key < (*root)->TreeNode_info.key)//if the new key is smaller
            insert_info_by_key (&((*root)->left),  data); //call  with left child
        else        //if the new node's key is higer
           insert_info_by_key (&((*root)->right),  data);//call  with right child
    }//else

}
```