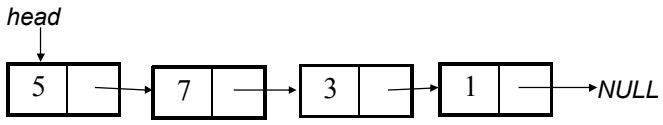
CSE 203: Data Structures and Algorithms-I

Linked Lists

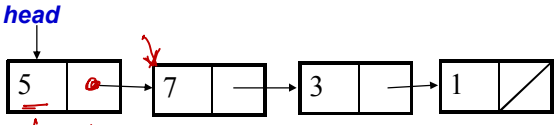


Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Linked List

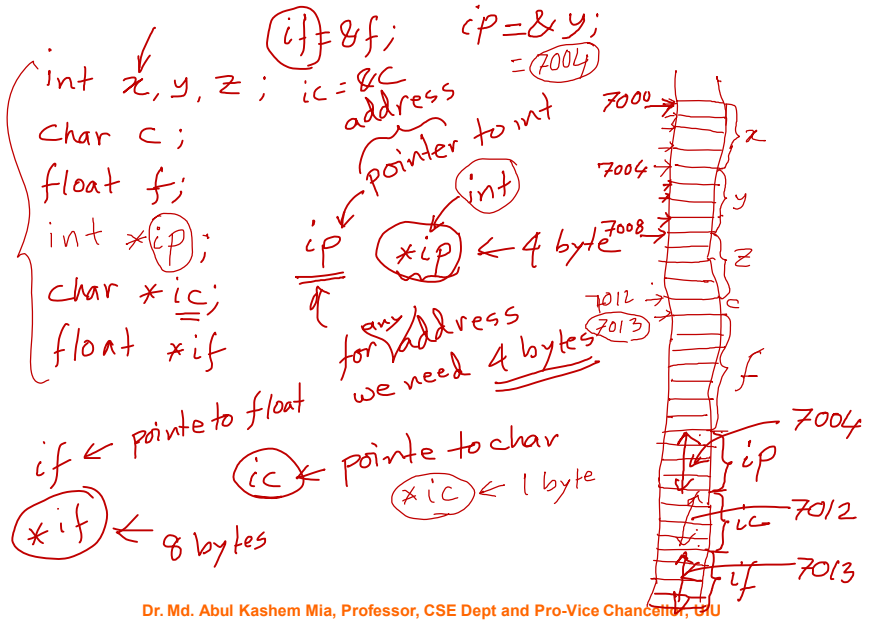
- A linked list is a linear collection of data elements (called nodes), where the linear order is given by means of pointers.
- Each node is divided into 2 parts:
  - 1st part contains the information of the element.
  - 2nd part is called the link field or next pointer field which contains the address of the next node in the list.

```
struct node {
    int value;
    struct node *next;
}
```

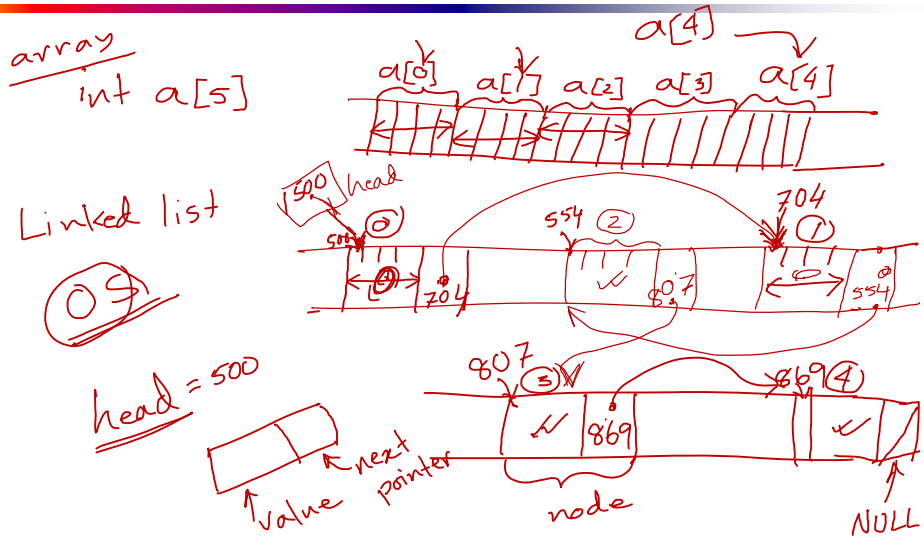


Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Linked List



Linked List

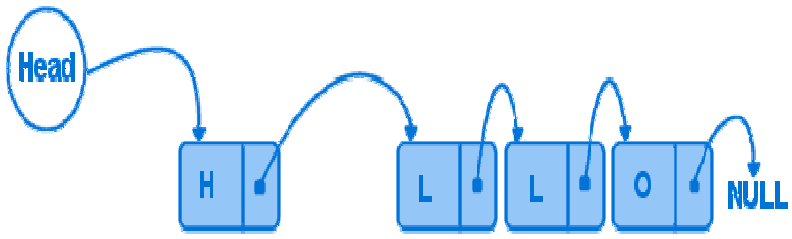


Basic Operations

- Insert: Add a new node in the first, last or interior of the list.
- Delete: Delete a node from the first, last or interior of the list.
- Search: Search a node containing particular value in the linked list.

Insertion to a Linear Linked list

- Add a new node at the first, last or interior of a linked list.

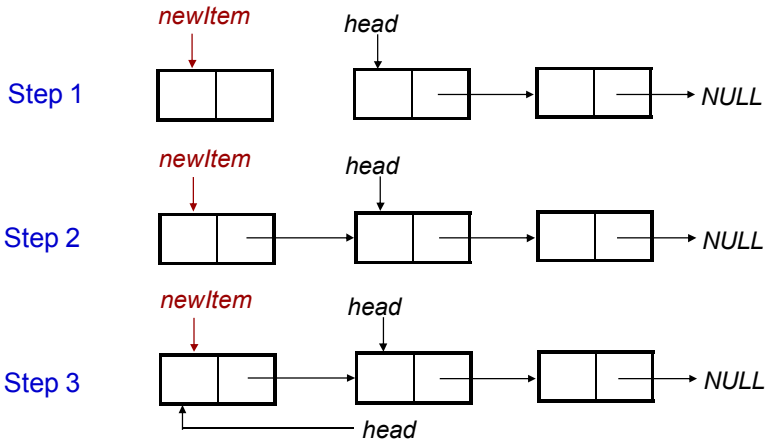


Insert First

- To add a new node to the head of the linear linked list, we need to construct a new node that is pointed by pointer *newitem*.
- Assume there is a global variable *head* which points to the first node in the list.
- The new node points to the first node in the list. The *head* is then set to point to the new node.

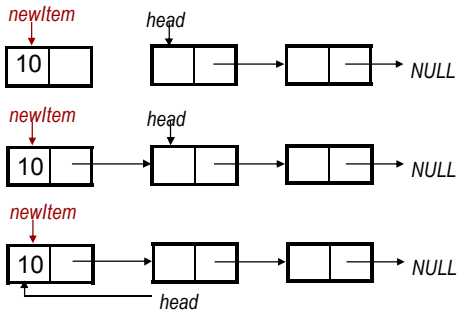
Insert First (Cont.)

- Step 1. Create a new node that is pointed by pointer *newItem*.
- Step 2. Link the new node to the first node of the linked list.
- Step 3. Set the pointer *head* to the new node.



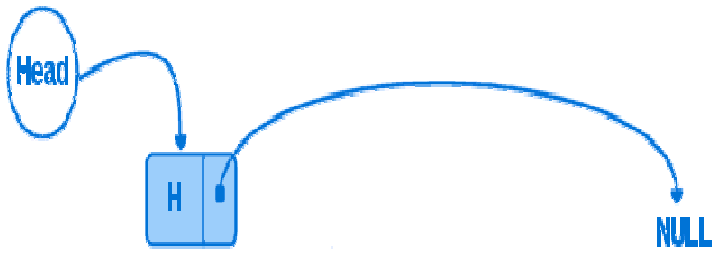
Insert First (Cont.)

```
struct node{
    int value;
    struct node *next;
};
struct node *head;

void insertHead(){
  //create a new node
  struct node *newItem;
  newItem=(struct node *)malloc(sizeof(struct node));
  newItem->value = 10;
  newItem->next = NULL;
  //insert the new node at the head
  newItem->next = head;
  head = newItem;
}
```



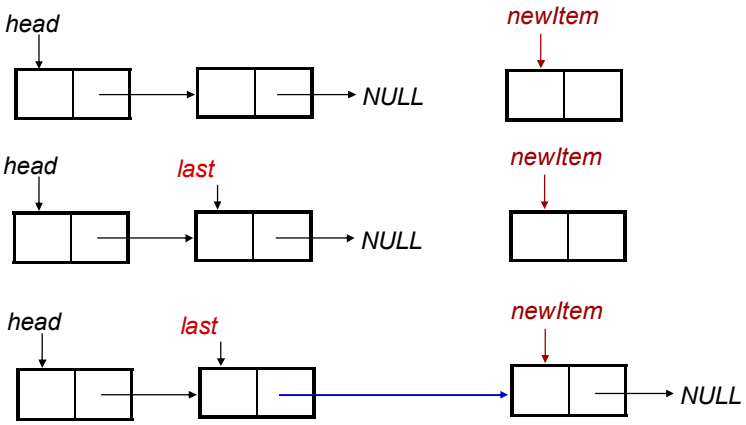
Insert Last

- To add a new node to the tail of the linear linked list, we need to construct a new node and set it's link field to “NULL".
- Assume the list is not empty, locate the last node and change it's link field to point to the new node.



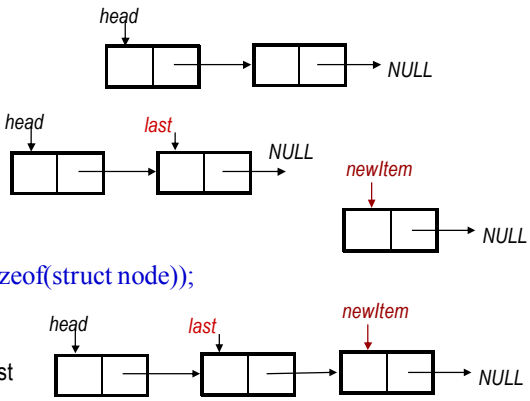
Insert Last (Cont.)

- Step1. Create the new node.
- Step2. Set a temporary pointer *last* to point to the last node.
- Step3. Set *last* to point to the new node.



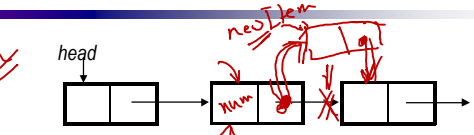
Insert Last (Cont.)

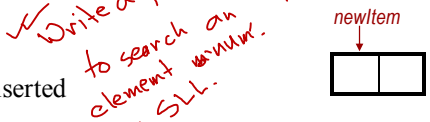
```
struct node{
    int value;
    struct node *next;
};
struct node *head;

void insertTail(){
  //create a new node to be inserted
  struct node *newItem;
  newItem=(struct node *)malloc(sizeof(struct node));
  newItem->value = 10;
  newItem->next = NULL;
  // set last to point to the last node of the list
  struct node *last = head;
  while (last->next != NULL)
     last = last->next;
  last->next = newItem;
}
```

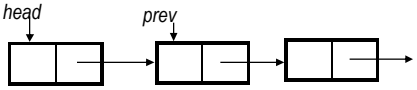


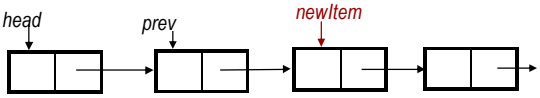
Insert Middle (after a desired node)

```
struct node{
    int value;
    struct node *next;
};
struct node *head;

void insertMiddle(int num){
  //create a new node to be inserted
  struct node *newItem;
  newItem=(struct node *)malloc(sizeof(struct node));
  newItem->value = 10;
  newItem->next = NULL;
  // set prev to point to the desired node of the list
  struct node *prev = head;
  while (prev->value != num){
     prev = prev->next;
  }
  newItem->next = prev->next;
  prev->next = newItem;
}
```







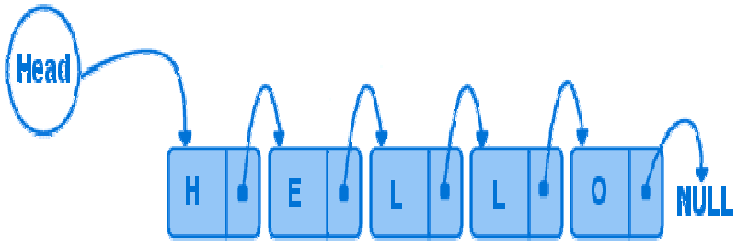


Printing List

```
void printList()
{
 if (head = = NULL)   // no list at all
     return;
 struct node *cur =  head;
 while (cur != NULL)
 {
     printf("%d \t", cur->value);
   cur = cur->next;
 }
}
```

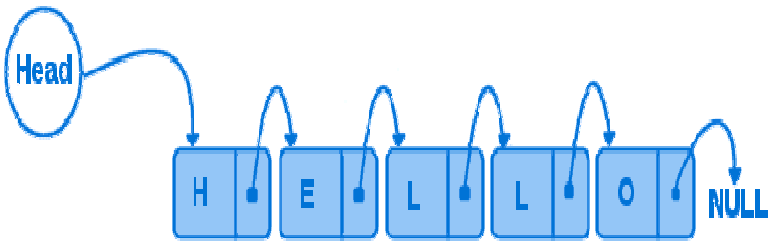
Deletion from a Linear Linked list

- Deletion can be done
  - At the first node of a linked list.
  - At the end of a linked list.
  - Within the linked list.



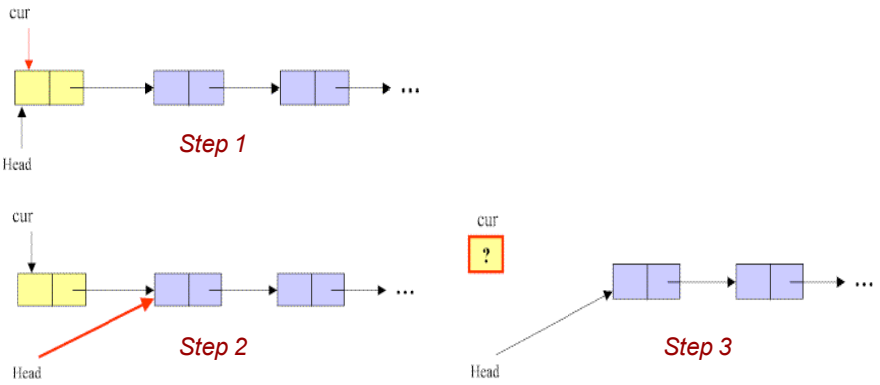
Delete First

- To delete the first node of the linked list, we not only want to advance the pointer *head* to the second node, but we also want to release the memory occupied by the abandoned node.



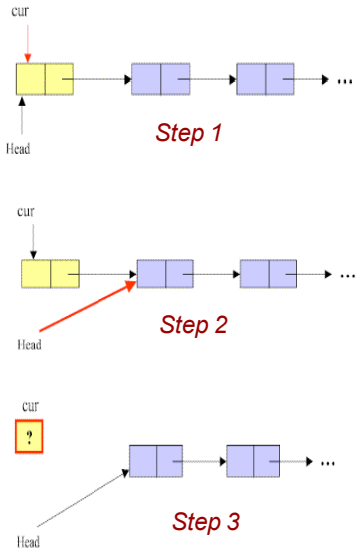
Delete First (Cont.)

- Step1. Initialize the pointer *cur* point to the first node of the list.
- Step2. Move the pointer *head* to the second node of the list.
- Step3. Remove the node that is pointed by the pointer *cur*.



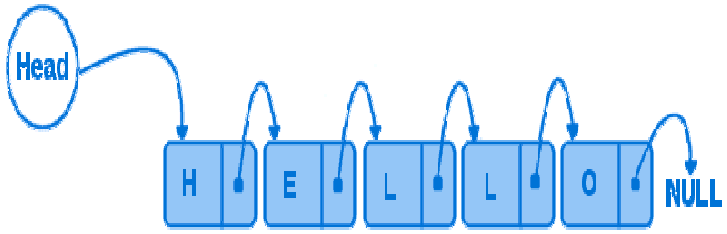
Delete First (Cont.)

```
void deleteHead()
{
  struct node *cur;
  if (head = = NULL)  //list empty
      return;
  cur = head;  // save head pointer
  head = head->next; //advance head
  free(cur);
}
```



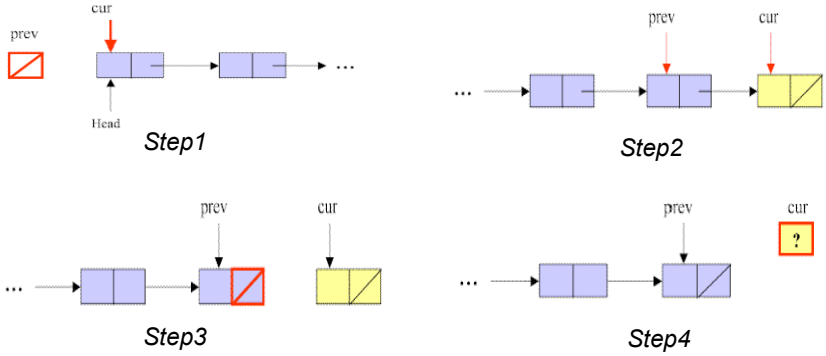
Delete Last

- To **delete** the last node in a linked list, we use a local variable, *cur*, to point to the last node. We also use another variable, *prev*, to point to the second last node in the linked list.



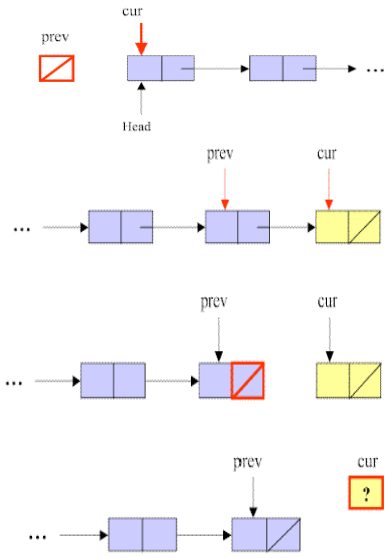
Delete Last (Cont.)

- Step1. Initialize pointer *cur* to point to the first node of the list, while the pointer *prev* has a value of NULL.
- Step2. Traverse the entire list until the pointer *cur* points to the last node of the list.
- Step3. Set NULL to *next* field of the node pointed by the pointer *prev*.
- Step4. Remove the last node that is pointed by the pointer *cur*.



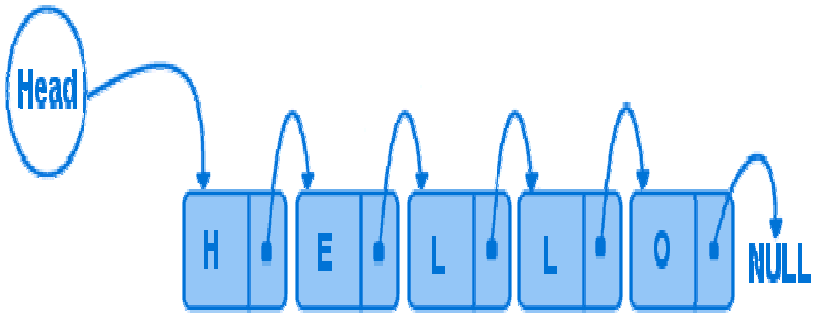
Delete Last (Cont.)

```
void deleteTail(){
  if (head = = NULL)  //list empty
      return;
  struct node *cur = head;
  struct node *prev = NULL;
  while (cur->next != NULL){
      prev = cur;
      cur=cur->next;
  }
  if (prev != NULL)
      prev->next = NULL;
  free(cur);
}
```



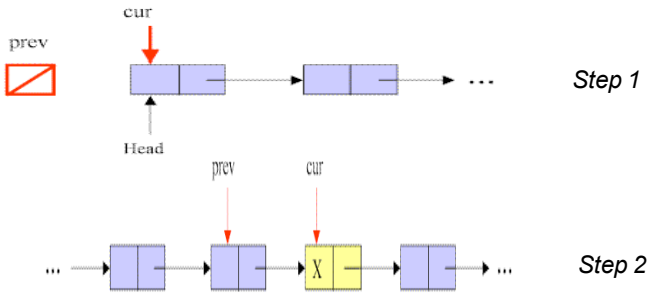
Delete Any

- *To **delete** a node that contains a particular value x in a linked list, we use a local variable, cur, to point to this node, and another variable, prev, to hold the previous node.*



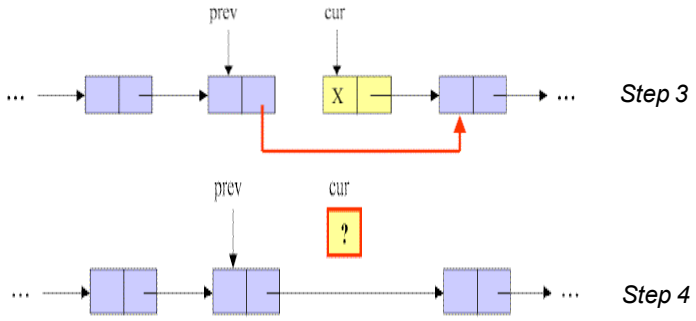
Delete Any (Cont.)

- Step1. Initialize pointer *cur* to point to the first node of the list, while the pointer *prev* has a value of null.
- Step2. Traverse the entire list until the pointer *cur* points to the node that contains value of x, and *prev* points to the previous node.
- ……..



Delete Any (Cont.)

- …….
- Step3. Link the node pointed by pointer *prev* to the node after the *cur*'s node.
- Step4. Remove the node pointed by *cur*.



Delete Any (Cont.)

```
void deleteAny( int x ){
  if (head = = NULL)   //list empty
      return;
  struct node *cur = head;
  struct node *prev = NULL;
  while (cur->value != x){
      prev = cur;
      cur=cur->next;
  }
 if (prev != NULL)
      prev->next = cur->next;
 free(cur);
}
```