Linked Lists and ADT

CS19001: Programming and Data Structures Laboratory

25-Oct-2019

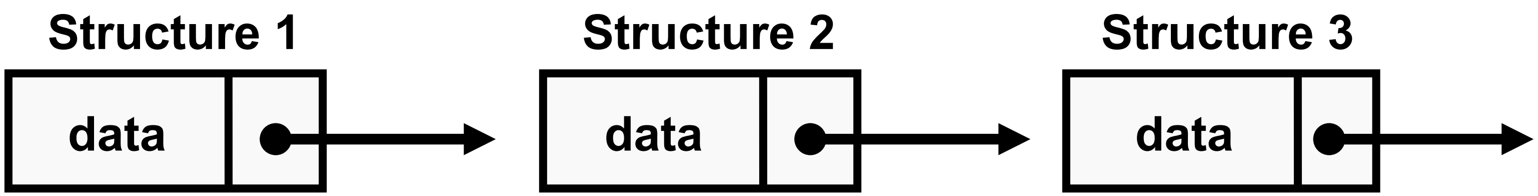
Aritra Hazra

Department of Computer Science & Engineering,
Indian Institute of Technology Kharagpur,
Paschim Medinipur, West Bengal, India – 721302.

http://cse.iitkgp.ac.in/~aritrah/course/lab/PDS/Autumn2019/

Lists

- A list refers to a sequence of data items
  - Example: An array
    - Array index is used for accessing and manipulating array elements
  - Problems with arrays
    - Array size specified at the beginning (at least during dynamic allocation)
      - realloc can be used to readjust size in middle, but contiguous chunk of memory may not be available
    - Deleting / Inserting an element may require shifting of elements
    - Wasteful of space
- A completely different way to represent a list (Linked List)
  - Make each data in the list part of a self-referential structure
  - The structure also contains a pointer or link to the structure (of the same type) containing the next data

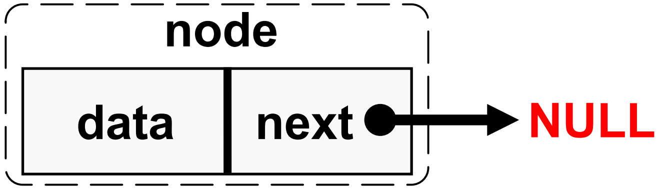


Single Linked Lists

- Let each structure of the list (lets call it node) have two fields:
  - One containing the data
  - Other containing address of the structure holding next data in the list
- The structures in the linked list need not be contiguous in memory
  - Ordered by logical links stored as part of data in the structure itself
  - The link is a pointer to another structure of the same type
- The pointer variable next contains either the address of the location in memory of the successor list element or the special value NULL
  - NULL is used to denote the end of the list (no successor element)

Definition of a Node:

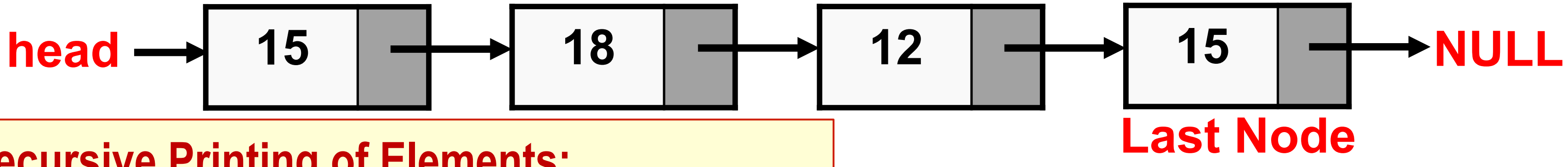
```
typedef struct node {
  int data;
  struct node *next;
} llNode;
llNode *head, *prev, *cur;
```



Creation of a Node:

```
llNode *createNode(int item)
{
  llNode *new = (llNode *)malloc(sizeof(llNode));
  if(new ==NULL) printf(“Malloc Error!”);
  else {
    new->data = item;  new->next = NULL;
  }
  return (new);
}
```

Traversal of Linked Lists



Recursive Printing of Elements:

```
void recPrintLL(llNode *head) {
   if(head != NULL){
      printf(“%d, ”, head->data);
      recPrintLL(head->next);
      printf(“%d, ”, head->data);
   }
}
```

Forward Printing

15, 18, 12, 15,

Backward Printing

15, 12, 18, 15,

Pointing Last Node in Single Linked List:

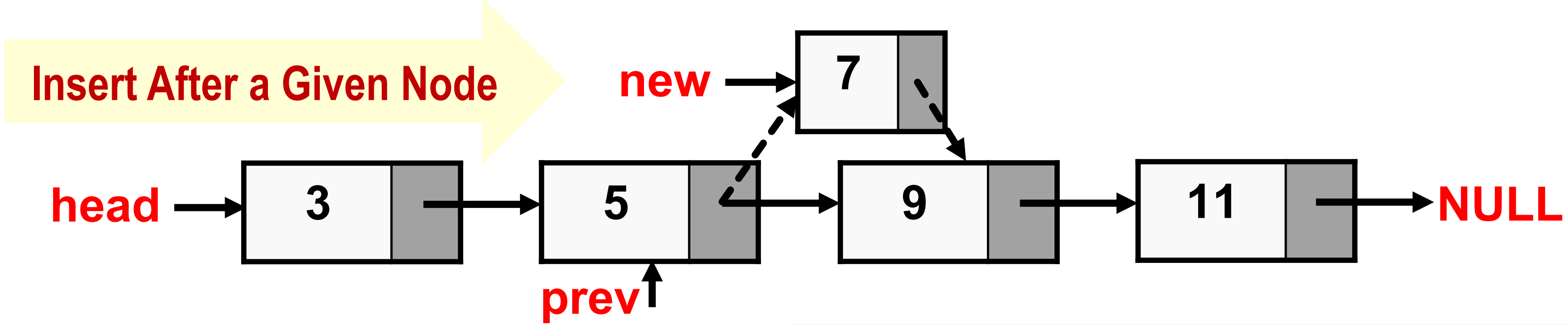
```
llNode *lastNodeLL(llNode *head)
{
   llNode *cur = head;
   if(cur != NULL) // no node
      while(cur->next != NULL)
         cur = cur->next;
   return (cur);
}
```

Finding an Element in Single Linked List:

```
llNode *searchLL
     (llNode *head, int elm)
{
   llNode *cur = head;
   while(cur != NULL) {
      if(cur->data == elm)
         break;
      cur = cur->next;
   }
   return (cur);
}
```

Need to Traverse all
N elements in list

Insertion into Linked Lists

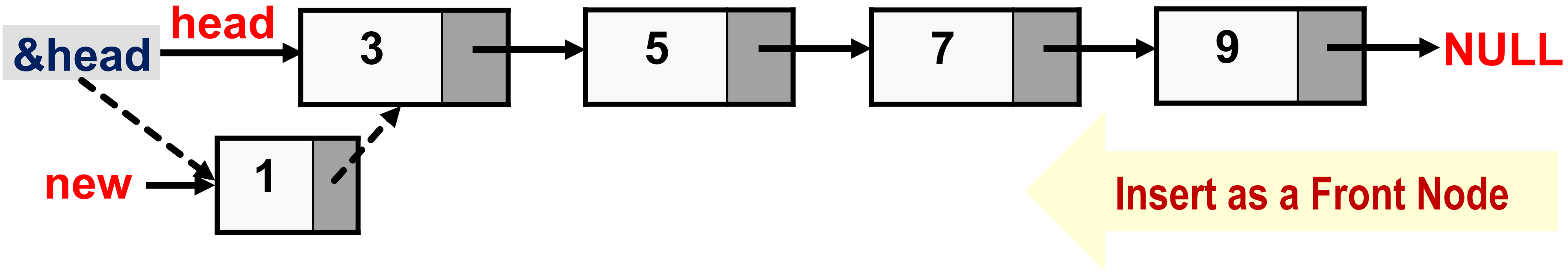


Insert a `new` Node after `prev` Node:

```
void insertAfterLL
   (llNode *prev, llNode *new)
{
   new->next = prev->next;
   prev->next = new;
}
```

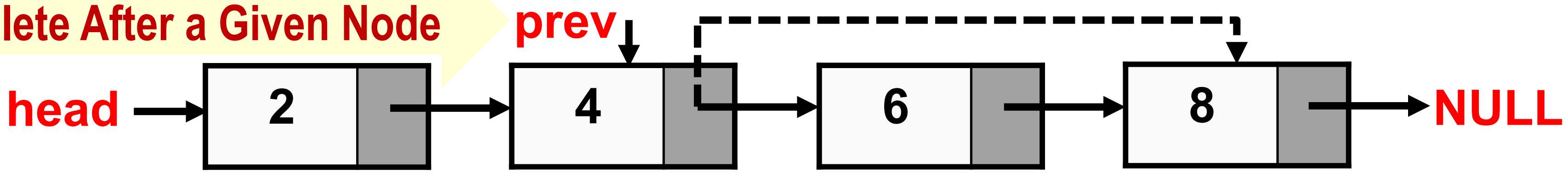
Insert a `new` Node in Front:

```
void *insertFrontLL
   (llNode **phead, llNode *new)
{
   new->next = (*phead);
   (*phead) = new;
}
```



Deletion from Linked Lists

Delete After a Given Node



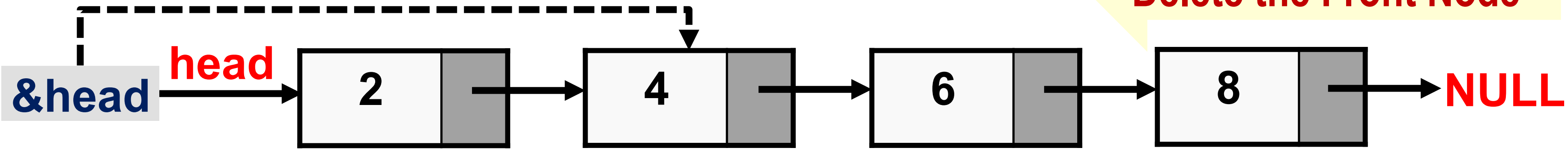
Delete a Node after `prev` Node:

```
void deleteAfterLL(llNode *prev)
{
   if(prev->next != NULL)
      prev->next =
         (prev->next)->next;
}
```

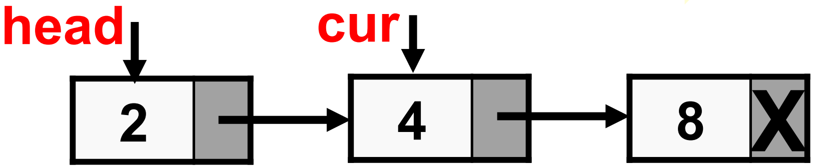
Delete the Front Node:

```
void *deleteFrontLL(llNode **phead)
{
   if((*phead) != NULL)
      (*phead) = (*phead)->next;
}
```

Delete the Front Node



Delete Random (Current) Node

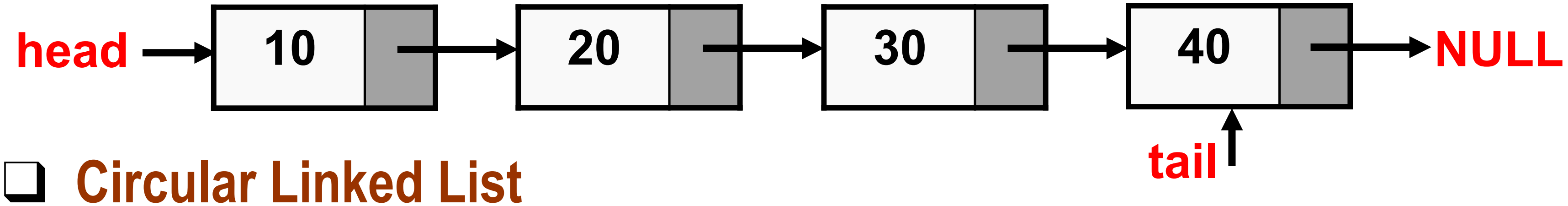


```
llNode *deleteCurLL(llNode **phead)
{
   cur->data = (cur->next)->data;
   deleteAfterLL(cur);
}
```

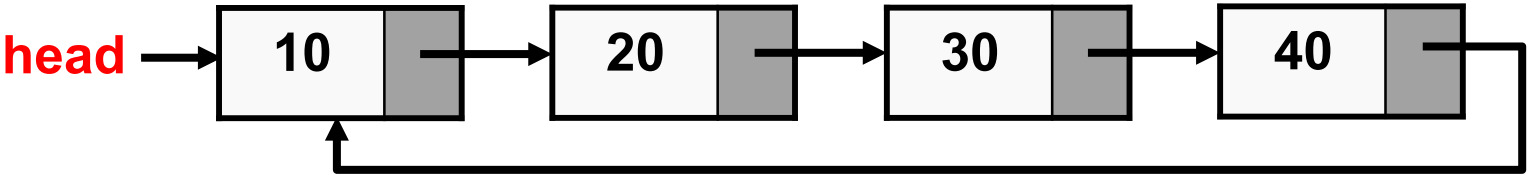
No need to traverse entire list (except `cur` being Last Node!)

Variations of Linked Lists

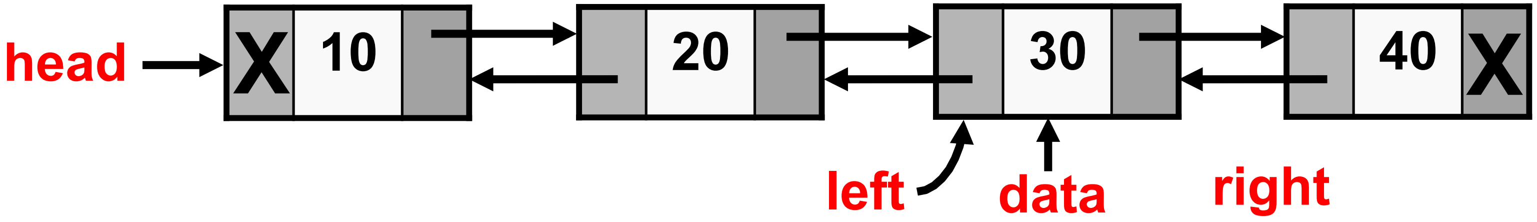
- Single Linked List with Head and Tail Pointers



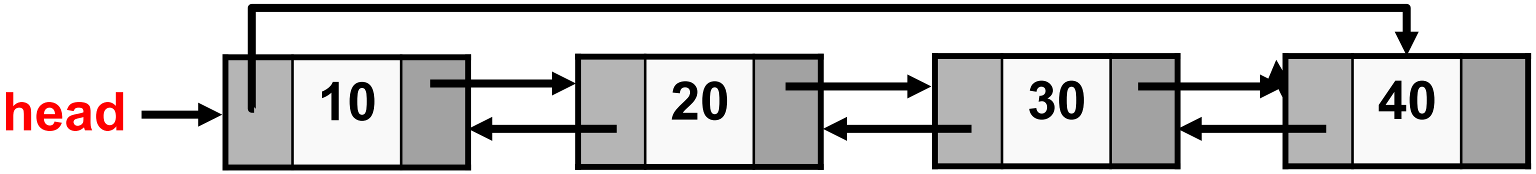
- Circular Linked List



- Double Linked List



- Circular Double Linked List



Thank You!