Data Structures and Algorithms
(ES221)

Linked List

Dr. Zubair Ahmad

Linked List

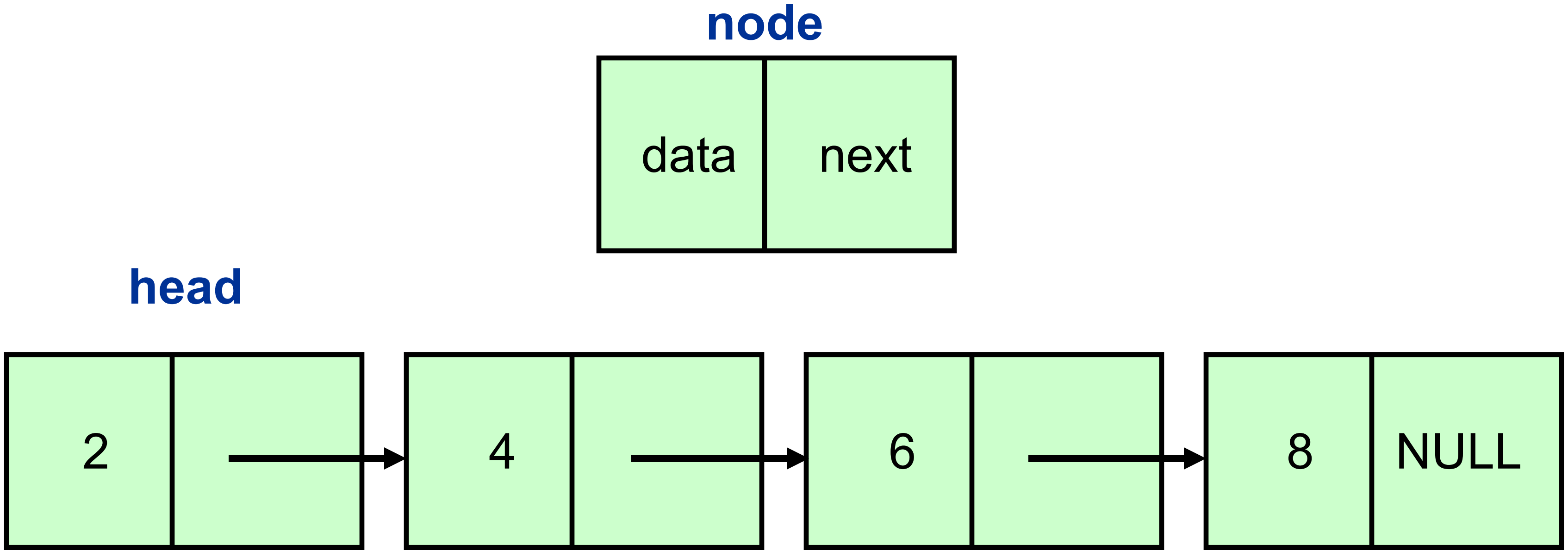
A linear data structure in which elements (nodes) are stored in memory **non-contiguously**

Each node consists of at least two types of elements

- An element to store some data/information
- A pointer to the next node in the list

- Usually the first node is known as the **header node**
- The pointer for the last node is usually a null pointer

Linked List



node

data next

head

2 → 4 → 6 → 8 NULL

Linked List

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

`int data;`

- This is an integer variable that stores the actual value of the node.

`node *next`

- This is a pointer to another node. It is used to link the current node to the next node in the list.
- If a node is the last node in the list, next will be NULL

- **head** is a **global pointer** of type **node***
- .It is used to keep track of the first node in the linked list.
- If the list is empty, head is set to NULL.

Linked List: Create the head node

```
struct node
     {
     int data;
     node *next;
     };
node *head;
int main()
     {
     head =
newnode;
     }
```

Linked List: Create the head node (Example)

```
#include <iostream>
using namespace std;

struct node
{
    int data;
    node *next;
};

node *head;

int main()
{
head = new node();

head->data = 10;
head->next = NULL;

cout << "Data in head node: " << head->data <<
endl;

    return 0;
}
```

Linked List: Add an element at the head node

Approach 1:

head node may contain data as well as serves as a global pointer to the start of the linked list

```
head = new node;
cin>>head->data;
head->next=NULL;
```

Linked List: Add an element at the head node

Approach 2:
head node only serves as a global pointer to the start of the linked list
head node does not contain any data

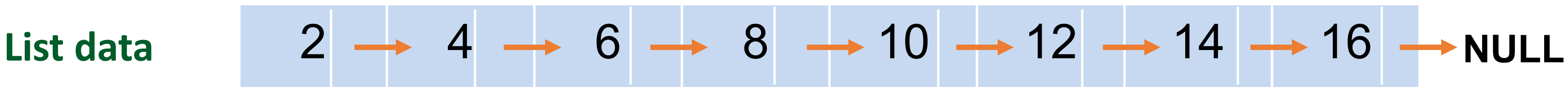
```
head->next = new node;
cin>> head->next ->data;
head->next->next=NULL;
```

OR

```
 node *ptr =new node;
     cin>> ptr ->data;
head->next=ptr;
ptr->next=NULL;
```

Linked List: Traverse through the linked list

- Visit each element of the list
  - e.g., print each element of the list on the screen



List data

2 → 4 → 6 → 8 → 10 → 12 → 14 → 16 → NULL

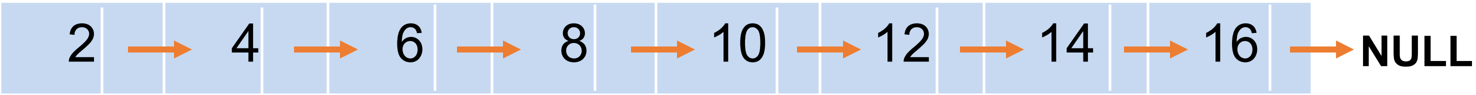
```
for (node *ptr = head; ptr != NULL; ptr = ptr->next)
    cout<< ptr->data;
```

Complexity : O(N)

Linked List: Search for an element in the list

- Search for a specific element in the list
  - e.g., search and return the pointer to the element containing data=10, if found in the list

List data



```
 SearchElement = 10;
for (node *ptr = head; ptr != NULL; ptr = ptr->next)
        {
        if (ptr->data == SearchElement){
            indexPtr = ptr;
             return indexPtr;
         }
}
 indexPtr = NULL;
 cout<<  “Element not found”  ;
 return indexPtr;
```

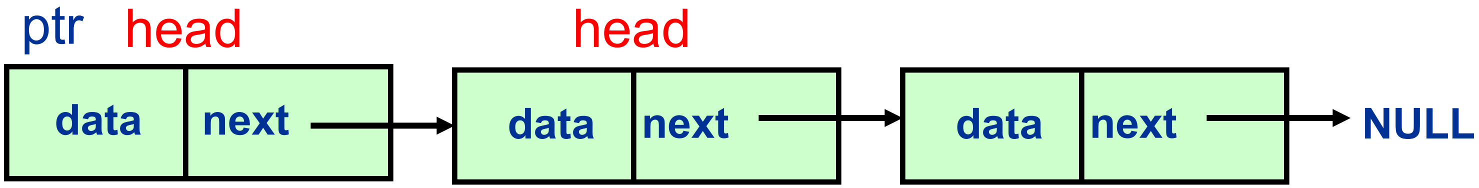
Complexity : $O(N)$

Linked List: Append a new element before/at the head node

- head node contains data as well as a global start pointer

```
node *ptr = new node;
 cin>> ptr ->data;
 ptr->next = head;
  head = ptr;
```

Complexity : O(*1*)



ptr head

head

data next

data next

data next

NULL

Linked List: Append a new element before/at the head node

```
head ---> [ 10 | * ] ---> [ 20 | * ] ---> [ 30 | NULL ]
```

User enters 5

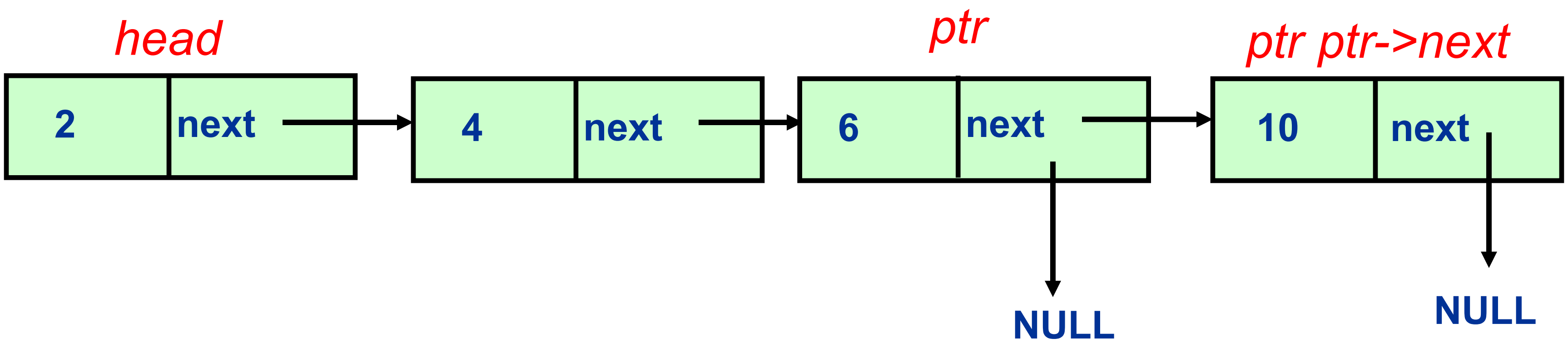
- A **new node** is created (ptr).
- ptr->data = 5.
- ptr->next = head (points to 10).
- head = ptr (head now points to 5).

```
head ---> [ 5 | * ] ---> [ 10 | * ] ---> [ 20 | * ] ---> [ 30 | NULL ]
```

Linked List: Append a new element after the tail node

- head node contains data as well as a global start pointer

```
node *ptr;
for (ptr = head; ptr->next != NULL; ptr = ptr->next){}
    ptr->next = new node;
      ptr = ptr->next;
      ptr->data = NewData; // e.g NewData =10;
      ptr->next = NULL;
```



head

2 next

4 next

ptr

6 next

NULL

ptr ptr->next

10 next

NULL

Complexity : O(*N*)

Example

```
head -> [10 | *] -> [20 | *] -> [30 | NULL]
```

Initialization:

- ptr = head → ptr points to the first node (10)

.

For Loop:

- ptr = ptr->next → ptr now points to 20.
- ptr = ptr->next → ptr now points to 30.
- **Loop Ends**: The next pointer of 30 is NULL, so the loop exits.

Example

Allocating New Node:

- ptr->next = new node; → A new node is created, and ptr->next (which was pointing to NULL) now points to this new node.

```
head -> [10 | *] -> [20 | *] -> [30 | *] -> [new node | NULL]
```

Move ptr to New Node:

- ptr = ptr->next; → Now ptr points to the newly created node.

Set New Data:

- ptr->data = NewData; → For example, if NewData = 10, the new node’ s data is set to 10.

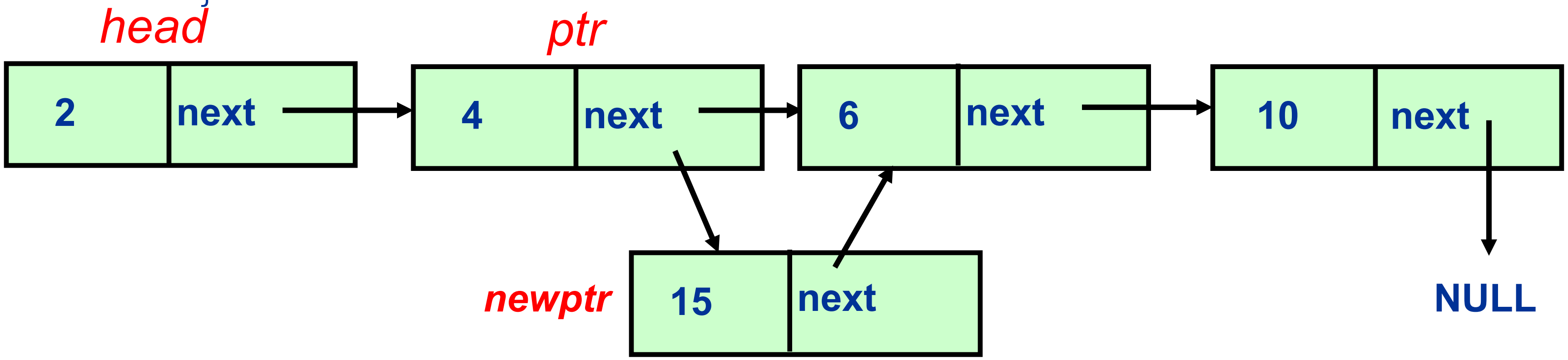
```
head -> [10 | *] -> [20 | *] -> [30 | *] -> [10 | NULL]
```

Linked List: Insert a new element after a specific node

- Suppose you want to insert after the node with data = ‘*afterValue*’

```
node* newptr;
for(node *ptr=head; ptr!=NULL;ptr=ptr->next){
  if(ptr->data == afterValue)    //  e.g afterValue =4;
    {
    newptr=new node;
    newptr->data=NewData;  //  e.g NewData =15;
    newptr->next=ptr->next;
    ptr->next=newptr;
    } // end if
 } // end for
```

Complexity : O(*N*)

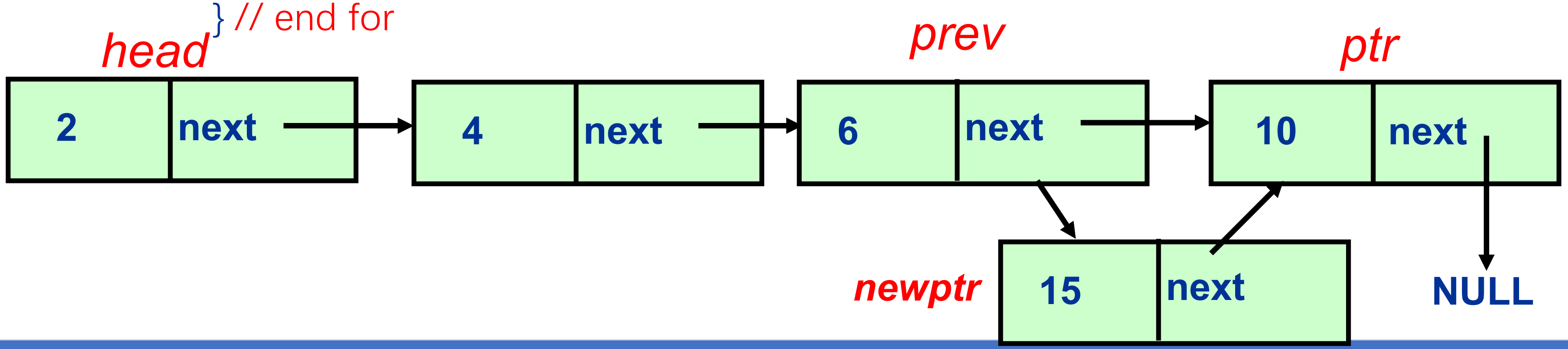


Linked List: Insert a new element before a specific node

- Suppose you want to insert before the node with data = '*beforeValue*'

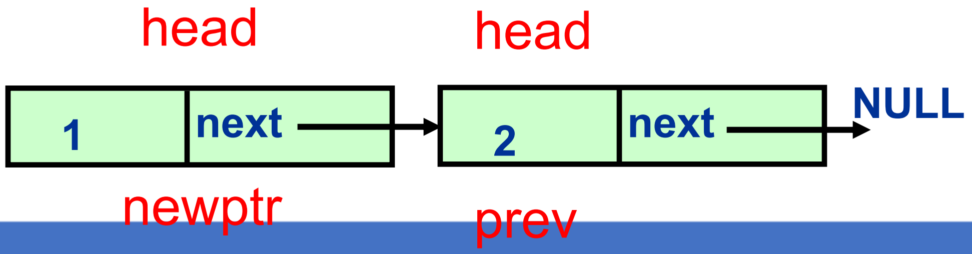
```
 node* newptr;
   node *prev=head;
   for(node *ptr=head->next; ptr!=NULL; ptr=ptr->next){
      if(ptr->data == beforeValue)    //  e.g beforeValue =10;
        {
        newptr=new node;
        newptr->data=NewData;     //  e.g NewData =16;
        prev->next=newptr;
        newptr->next=ptr;
        } // end if
      prev=prev->next;
    } // end for
```

Complexity : O(*N*)



Linked List: Insert a new element before a specific node (Only one node is currently there)

```
node* newptr;
 node *prev=head;
 if(head->data == beforeValue) {    // e.g beforeValue =2;
       newptr=new node;
       head=newptr;
       head->next=prev;
       newptr->data=NewData;    //   e.g NewData =1;
       return;
       }
```

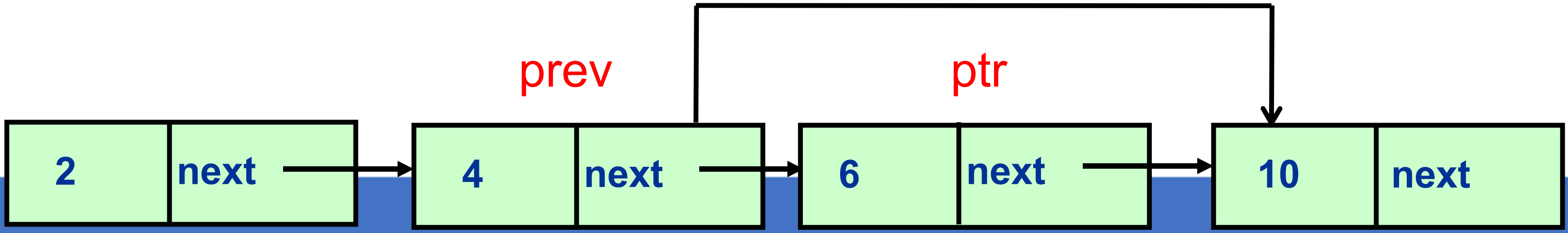


Linked List: Delete an element from the list

- Suppose you want to delete the node with data = '*ValueToDelete*'

```
node *prev=head;
for(node *ptr=prev->next; ptr!=NULL; ptr=ptr->next){
if(ptr->data == ValueToDelete) // e.g ValueToDelete = 6;
   {
   prev->next=ptr->next;
   delete(ptr);
   return;
   }
   prev=prev->next;
}
```

Complexity : O(*N*)

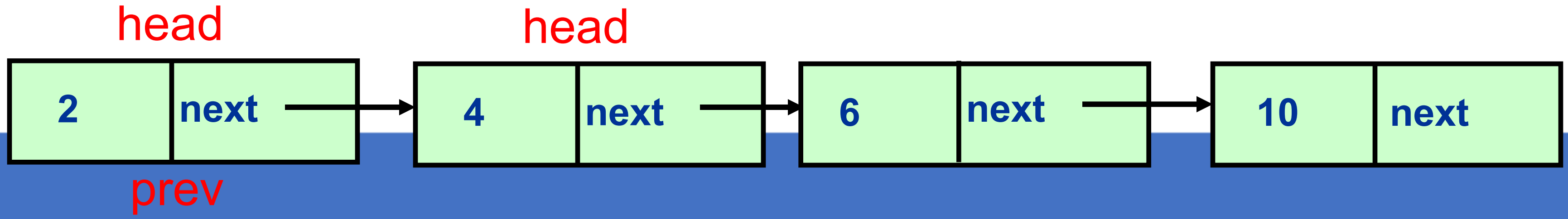


Linked List: Delete an element from the list

- **Delete the head node from the list**

```
node *prev=head;
 if(head->data == ValueToDelete)
  {
  head=head->next;
  delete(prev);
    return;
  }
```

Complexity : O(1)



Quiz!!

Quiz on next week Tuesday. Time and Venue will communicated later

Questions?

zahmaad.github.io