DATA STRUCTURES AND ALGORITHMS

Linked List

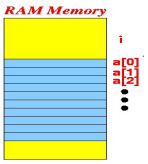
By
Zainab Malik

Content

- Problems of Arrays
- Introduction Linked List
  - Properties of Linked List
  - Operations of Linked List
  - Advantages/Disadvantages of Linked List
  - Applications of LinkedList

Arrays-Problem

- It occupies consecutives slots in memory



- Once the size of the array is defined, either for static or dynamic array, it cannot be modified later on.

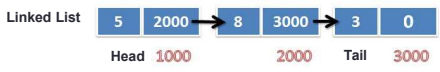
Link List

- A linked list is a linear collection of data elements, called nodes. The linear order is given by mean of pointers. Node is a structure that can be divided into two or more parts.
- It can be of following types:
  - Linear Linked list or one way list
  - Doubly Linked list or two way list
  - Circular Linked list
  - Header Linked list
  - Two-way header list

Linear Linked List

- In a linear linked list, also called singly linked list or one-way list, each node is divided into two parts
  - First part contains the information/content/data
  - Second part contains the address of next node or the pointer to next node

  - The first node of a linear linked list is represented by a pointer, called head that stores the address of first node while the last node is represented by another pointer, called tail that stores the address of last node.



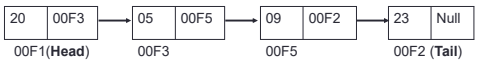
Operations on Linear Linked List

- Traversing
- Searching
- Insertion
  - AddToHead
  - AddToTail
  - AddAfterGivenElement
  - AddBeforeGivenElement
- Removal
  - RemoveFromHead
  - RemoveFromTail
  - RemoveGivenItem

7

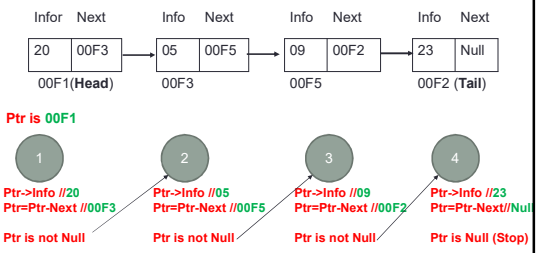
Traversing(list)

1. Set a pointer ptr = head (starting node)
2. Repeat step 3-4 while (ptr is not Null)
3. Print the info of ptr (cout<<ptr->info)
4. Set the ptr to its next (ptr=ptr->next)
5. EndWhile
6. Exit



8

Traversing(list)

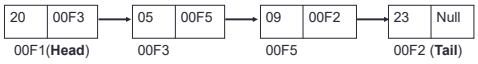


Ptr is 00F1

9

totalElements(list)

1. Set a pointer ptr = head (starting node)
2. Set a counter =1
3. Repeat step 3-4 while (ptr is not Null)
4. increment counter by one
5. Set the ptr to its next (ptr=ptr->next)
6. EndWhile
7. Print counter
8. Exit



10

Searching (list, item)

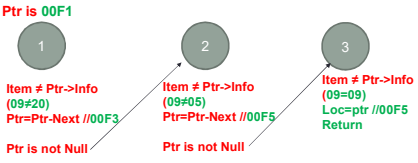
1. Set ptr=head
2. Set loc=0
3. Repeat step 3 and 4 While (ptr is not Null)
4. If (item==ptr->info) then
5. Set loc=ptr
6. Return loc
7. EndIf
8. Set ptr=ptr->next
9. EndWhile
10. Return loc

11

Searching(list,item)

item=09

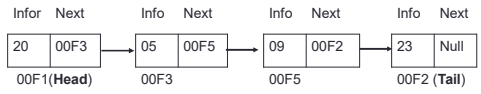
Ptr is 00F1



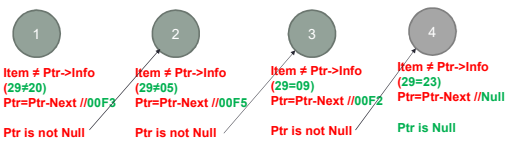
12

Searching(list,item)

item=29



Ptr is 00F1



13

Insertion-AddtoHead(list, element)

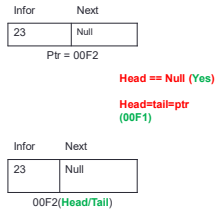
1. Set ptr=address of newly constructed Node
2. Set ptr->info=element
3. Set ptr->next=Null
4. If (head=Null) then
5. Set head=tail=ptr
6. else
7. Set ptr->next=head
8. set Head=ptr
9. Endif
10. Exit

14

addToHead(list,element)

element=23

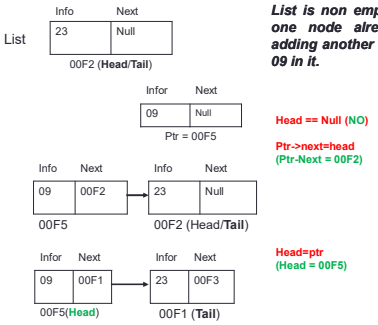
List is empty initially and we are adding a node with element 23 in it.



15

addToHead(list,element)

element=09

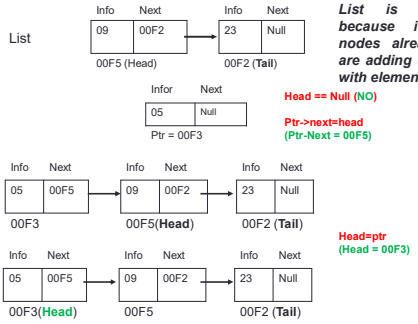


List is non empty because it has one node already and we are adding another node with element 09 in it.

16

addToHead(list,element)

element=05

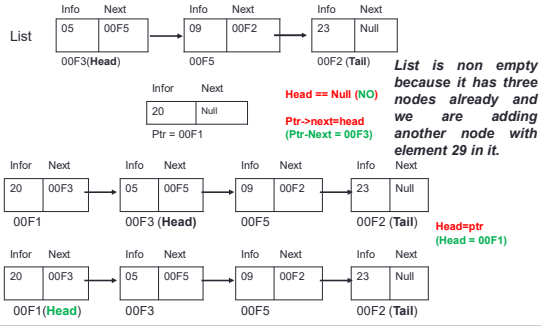


List is non empty because it has two nodes already and we are adding another node with element 05 in it.

17

addToHead(list,element)

element=29

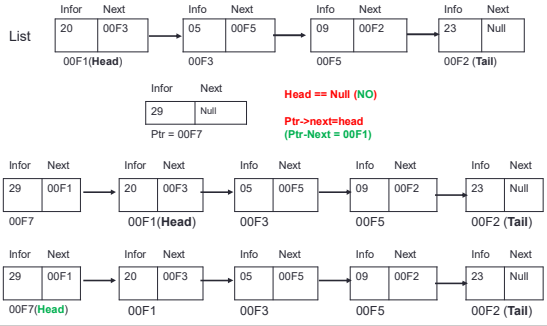


List is non empty because it has three nodes already and we are adding another node with element 29 in it.

18

addToHead(list,element)

element=29



19

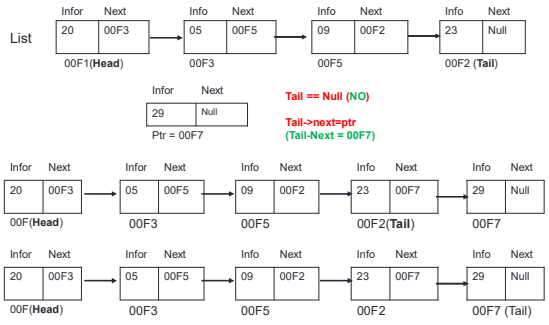
Insertion-AddtoTail(list, element)

1. Set ptr=address of newly constructed Node
2. Set ptr->info=element
3. Set ptr->next=Null
4. If (tail=Null) then
5. Set head=tail=ptr
6. else
7. set tail->next=ptr
8. set tail=ptr
9. Endif
10. Exit

20

addToTail(list,element)

element=29



21

Operations on Linear Linked List

- Traversing
- Searching
- Insertion
  - AddToHead
  - AddToTail
  - AddAfterGivenElement
  - AddBeforeGivenElement
- Removal
  - RemoveFromHead
  - RemoveFromTail
  - RemoveGivenItem

22

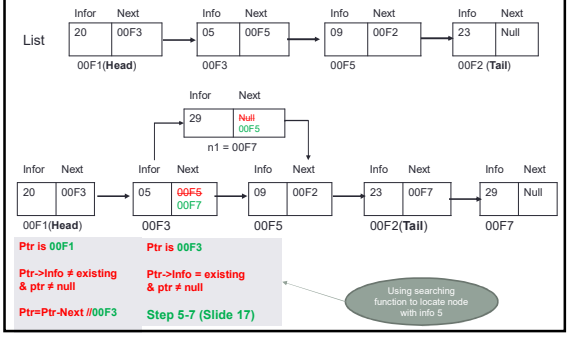
Insertion-AddAfter (list,existing, element)

- Case1:List is Empty
  - Display message (“there is no existing element in the list”)
- Case 2:List is not empty and the exiting is found at tail
  - Call addToTail function
- Case 3:List is not empty and existing can be somewhere after head
  1. Create a new node as n1
  2. Call searching function and save return address as loc
  3. If(loc==null)
     Display message (“existing not found”)
     Else
     Set ptr->next=loc->next
     Set loc->next=ptr

23

Insertion-AddAfter (list,existing, element)

existing=05, element=29 (Illustrating Case 3 only)



24

Operations on Linear Linked List

- Traversing
- Searching
- Insertion
  - AddToHead
  - AddToTail
  - AddAfterGivenElement
  - AddBeforeGivenElement
- Removal
  - RemoveFromHead
  - RemoveFromTail
  - RemoveGivenItem

25

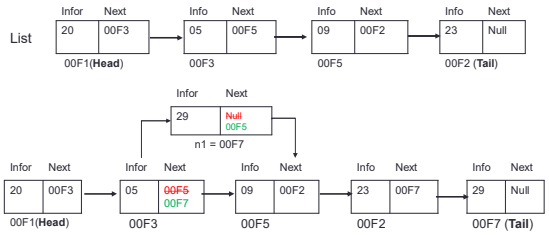
Insertion-AddBefore(list,existing, element)

- Case 1:List is Empty
  - Display message (“there is no existing element in the list”)
- Case 2:List is not empty and the exiting is found at head
  - Call addToHead function
- Case 3:List is not empty and existing can be somewhere after head
  1. Create a new node as n1
  2. Set ptr=head
  3. Repeat step 4 While (ptr!=null && ptr->next->info != existing)
  4. Ptr=ptr->next
  5. If(ptr==null)
     Display message (“existing not found”)
  6. Else if (ptr->next->info == existing)
     Set n1->next=ptr->next
     Set ptr->next=n1

26

Insertion-AddBefore (list,existing, element)

existing=09, element=29 (Illustrating Case 3 Only)



Ptr is 00F1

Ptr->next->Info ≠ existing & ptr ≠ null

Ptr=Ptr-Next //00F3

Ptr is 00F3

Ptr->next->Info = existing & ptr ≠ null

Step 5-7 (Slide 18)

27

Operations on Linear Linked List

- Traversing
- Searching
- Insertion
  - AddToHead
  - AddToTail
  - AddAfterGivenElement
  - AddBeforeGivenElement
- Removal
  - RemoveFromHead
  - RemoveFromTail
  - RemoveGivenItem

28

Operations on Linear Linked List

- Traversing
- Searching
- Insertion
  - AddToHead
  - AddToTail
  - AddAfterGivenElement
  - AddBeforeGivenElement
- Removal
  - RemoveFromHead
  - RemoveFromTail
  - RemoveGivenItem

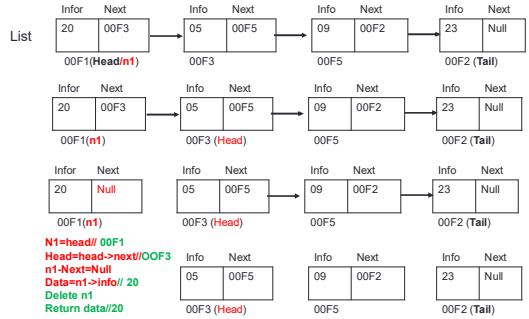
29

Deletion-RemoveFromHead(list)

- Case 1:List is Empty
  - Display message (“there is no existing element in the list to remove”)
- Case 2: If only one element
  1. Save info of head/tail in variable Data
  2. Delete head/tail
  3. Set head and tail both equal to 0
  4. Return Data
- Case 3:List is not empty
  1. Set n1=head
  2. Head=head->next
  3. Set n1->next=Null
  4. Save Info of n1 in a variable “Data”
  5. Delete n1
  6. Return Data

30

Deletion-RemoveFromHead(list) (Illustrating Case 3 Only)



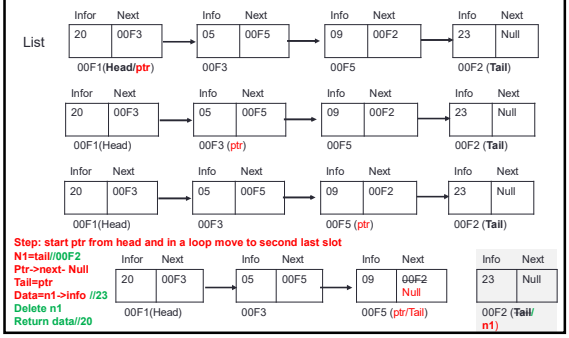
31

Deletion-RemoveFromTail(list)

- Case 1:List is Empty
  - Display message (“there is no existing element in the list to remove”)
- Case 2: If only one element
  1. Save info of tail/head in variable Data
  2. Delete tail/head
  3. Set head and tail both equal to 0
  4. Return Data
- Case 3:List is not empty
  1. Set ptr=head
  2. Repeat step 3 while (ptr->next != tail)
  3. Ptr=ptr->next
  4. Set n1=tail
  5. Ptr->next=Null
  6. Tail=ptr
  7. Save info of n1 in a variable Data
  8. Delete n1
  9. Return Data

32

Deletion-RemoveFromTail(list) (Illustrating Case 3 Only)



33

Deletion-RemoveGivenItem(list, item)

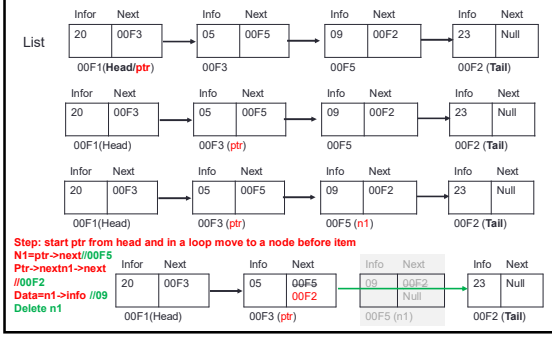
- Case 1:List is Empty
  - Display message (“there is no existing element in the list to remove”)
- Case 2: only one element and item exist at that single node
  - Delete head node
  - Reset head and tail as 0.
- Case 2: Element found at head
  - Call removeFromHead
- Case 3: Element found at tail
  - Call removeFromTail
- Case 4: Element may be Somewhere in between
  1. Set ptr=head
  2. Repeat step 3 while (ptr->next !=Null && ptr->next->info != item)
  3. Ptr=ptr->next
  4. If (ptr->next==Null)
     1. Display error message

     else
     1. Set n1=ptr->next
     2. Ptr->next=n1->next
     3. Delete n1

34

Deletion-RemoveGivenItem(list, item)(Illustrating Case 4 Only)

item=09



35

Advantages of Singular LinkedList

- **ADVANTAGE** :-

1. It does not need movement of elements for insertion and deletion.
2. Space is not wasted as we can get space according to our requirements.
3. Its size is not fixed.
4. It can be extended or reduced according to requirements.
5. Elements may or may not be stored in consecutive memory available
6. It is less expensive.

36

Disadvantages of Singular LinkedList

DISADVANTAGE :-

1. It requires more space as pointers are also stored with information.
2. Different amount of time is required to access each element.
3. If we have to go to a particular element then we have to go through all those elements that come before that element.
4. we can not traverse it from last node.
5. It is not easy to sort the elements stored in the linear linked list.

37

Applications

- Implementation of Stack and Queue
- Implementation of graphs : Adjacency list representation of graphs is most popular which uses linked list to store adjacent nodes/vertices.
- Dynamic memory allocation
- Performing arithmetic operations on long integers
- Manipulation of polynomials by storing constants in the node of linked list

38

Thank You