SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-(IOT Including Cyber Security &BCT)

COURSE NAME : 19ITT201- DATA STRUCTURES

II YEAR / III SEMESTER

Unit 1- LINEAR STRUCTURES AND TREES

Topic 4 : Linked List Based Implementation

Problem

Traversal

Elements or nodes traversal is difficult in linked list. We can not randomly access any element as we do in array by index. For example if we want to access a node at position n then we have to traverse all the nodes before it. So, time required to access a node is large.

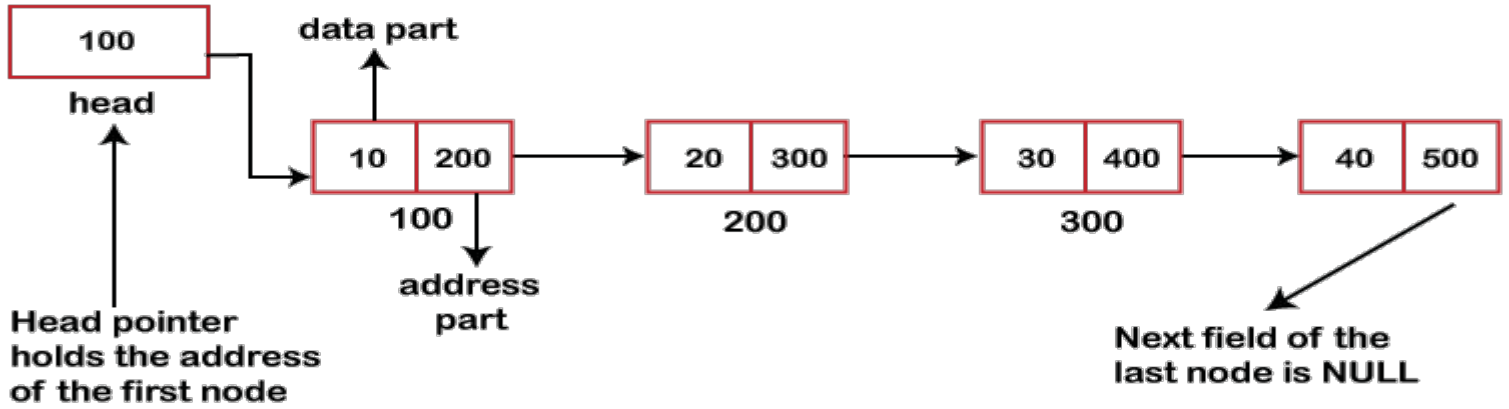
Reverse Traversing

In linked list reverse traversing is really difficult. In case of doubly linked list its easier but extra memory is required for back pointer hence wastage of memory.

Linked List Based Implementation

Define Linked List

- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.
- (Or) A linked list is a sequence of data structures, which are connected together via links.
- The elements in a linked list are linked using pointers as shown in the below



Types of Linked List

Following are the various types of linked list.

Simple or Singly Linked List − Item navigation is forward only.

Doubly Linked List − Items can be navigated forward and backward.

Circular Linked List − Last item contains link of the first element as next and the first element has a link to the last element as previous.

Linked List Based Implementation **–Cont..**

Operations on linked list

1.Creation of a list

2.Insertion of a list

3.Modification of a node

4.Deletion of node

5.Traversal of a list

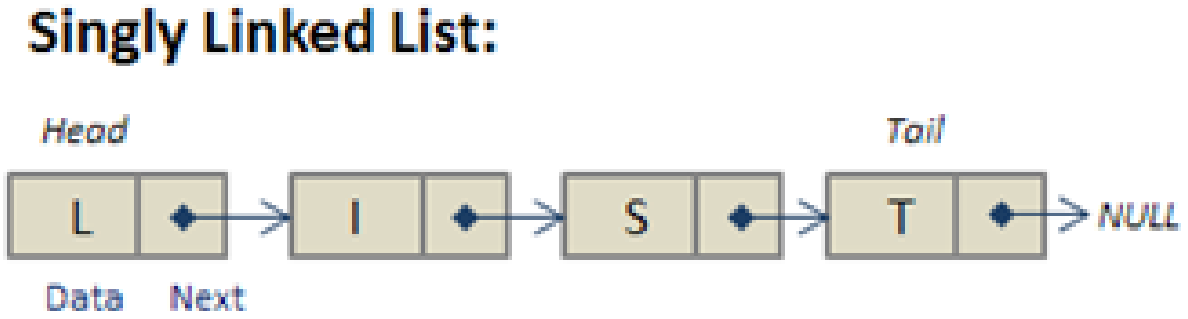
Singly Linked List Based Implementation

We also created a simple linked list with 3 nodes and discussed linked list traversal.

1.Creation of list

A linked list node

```
struct Node
{
  int data;
  struct Node *next;
};
```



Singly Linked List Based Implementation

In this post, methods to insert a new node in linked list are discussed. A node can be added in three ways

1) At the front of the linked list

2) After a given node.

3) At the end of the linked list.

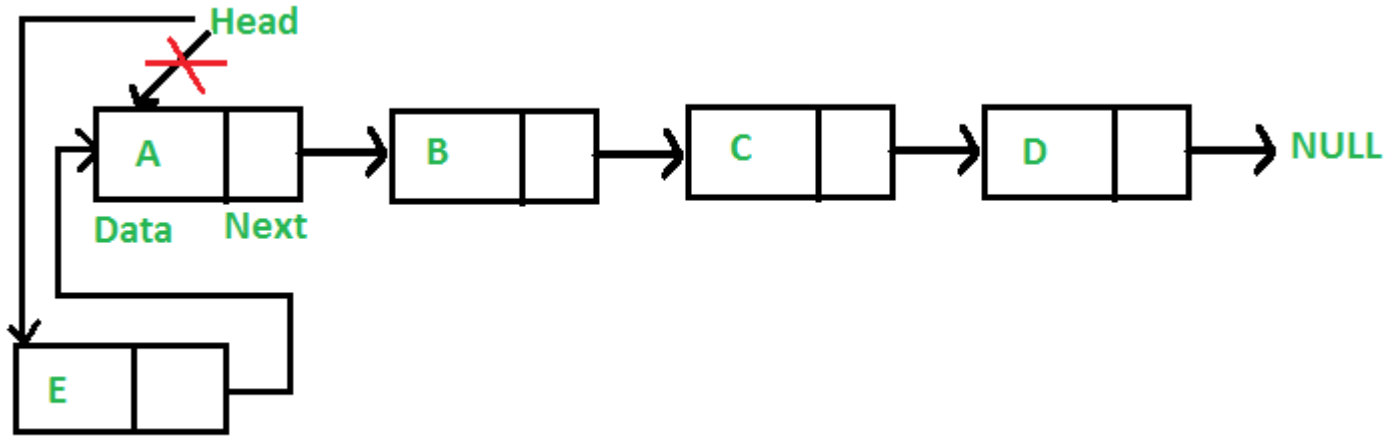
Singly Linked List Based Implementation –Cont..

At the front of the linked list

For example, if the given Linked List is 10->15->20->25 and we add an item 5 at the front, then the Linked List becomes 5->10->15->20->25.

Let us call the function that adds at the front of the list is push().

The push() must receive a pointer to the head pointer, because push must change the head pointer to point to the new node



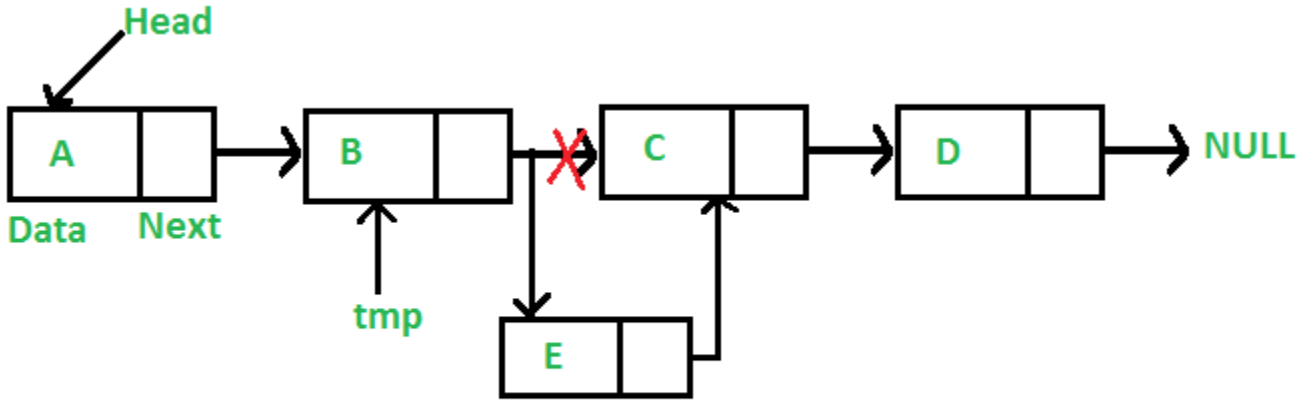
Singly Linked List Based Implementation –Cont..

```
void push(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node = (struct Node*) malloc(sizeof(struct
Node));

    /* 2. put in the data  */
    new_node->data  = new_data;

    /* 3. Make next of new node as head */
    new_node->next = (*head_ref);

    /* 4. move the head to point to the new node */
    (*head_ref)    = new_node;
}
```

Singly Linked List Based Implementation –Cont..

2) After a given node.

are given a pointer to a node, and the new node is inserted after the given no



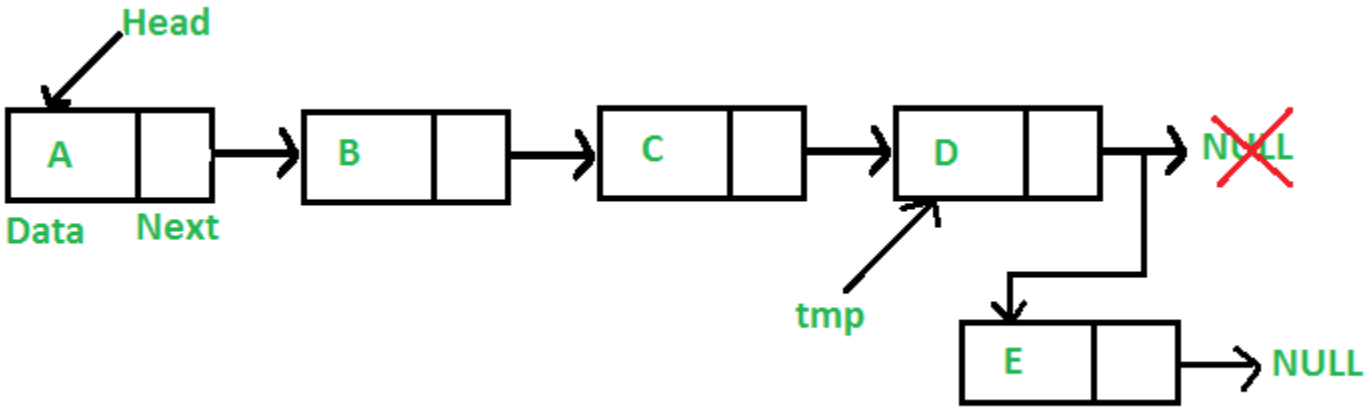
Singly Linked List Based Implementation –Cont..

```
void insertAfter(struct Node* prev_node, int new_data)
{
   /*1. check if the given prev_node is NULL */
   if (prev_node == NULL)
   {
   printf("the given previous node cannot be NULL");
   return;
   }

   /* 2. allocate new node */
   struct Node* new_node =(struct Node*) malloc(sizeof(struct Node));

   /* 3. put in the data */
   new_node->data = new_data;

   /* 4. Make next of new node as next of prev_node */
   new_node->next = prev_node->next;

   /* 5. move the next of prev_node as new_node */
   prev_node->next = new_node;
}
```

Singly Linked List Based Implementation –Cont..

3) At the end of the linked list.

- The new node is always added after the last node of the given Linked List. For example if the given Linked List is 5->10->15->20->25 and we add an item 30 at the end, then the Linked List becomes 5->10->15->20->25->30.
- Since a Linked List is typically represented by the head of it, we have to traverse the list till the end and then change the next to



Singly Linked List Based Implementation –Cont..

```
void append(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
    struct Node *last = *head_ref;  /* used in step 5*/
     /* 2. put in the data  */
    new_node->data  = new_data;
     /* 3. This new node is going to be the last node, so make next
          of it as NULL*/
    new_node->next = NULL;
     /* 4. If the Linked List is empty, then make the new node as head */
    if (*head_ref == NULL)
    {
      *head_ref = new_node;
      return;
    }
/* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
    last->next = new_node;
    return;
}
```

Activity

Advantages

- **Dynamic data structure:** A linked list is a dynamic arrangement so it can grow and shrink at runtime by allocating and deallocating memory. So there is no need to give the initial size of the linked list.
- **No memory wastage:** In the Linked list, efficient memory utilization can be achieved since the size of the linked list increase or decrease at run time so there is no memory wastage and there is no need to pre-allocate the memory.
- **Implementation:** Linear data structures like stack and queues are often easily implemented using a linked list.
- **Insertion and Deletion Operations:** Insertion and deletion operations are quite easier in the linked list. There is no need to shift elements after the insertion or deletion of an element only the address present in the next pointer needs to be updated.

Disadvantages

Memory usage: More memory is required in the linked list as compared to an array. Because in a linked list, a pointer is also required to store the address of the next element and it requires extra memory for itself.

Traversal: In a Linked list traversal is more time-consuming as compared to an array. Direct access to an element is not possible in a linked list as in an array by index. For example, for accessing a mode at position n, one has to traverse all the nodes before it.

Reverse Traversing: In a singly linked list reverse traversing is not possible, but in the case of a doubly-linked list, it can be possible as it contains a pointer to the previously connected nodes with each node. For performing this extra memory is required for the back pointer hence, there is a wastage of memory.

Random Access: Random access is not as possible in a linked list due to

Assessment 1

1. List out the advantages of linked list based implementation

a)_____

b)_____

c)_____

d)_____

2. Identify the disadvantages of linked list based implementation

a)_____

b)_____

c)_____

d)_____

REFERENCES

1. M. A. Weiss, “Data Structures and Algorithm Analysis in C”, Pearson Education, 8th Edition, 2007. [Unit I, II, III, IV,V]

2. A. V. Aho, J. E. Hopcroft and J. D. Ullman, “Data Structures and Algorithms”, Pearson Education, 2nd Edition, 2007 [Unit IV].

3. A.M.Tenenbaum, Y. Langsam and M. J. Augenstein, “Data Structures using C”,PearsonEducation, 1st Edition, 2003.(UNIT I,II,V)

4.https://www.youtube.com/watch?v=0xoYNbVTiSE&t=256s

THANK YOU