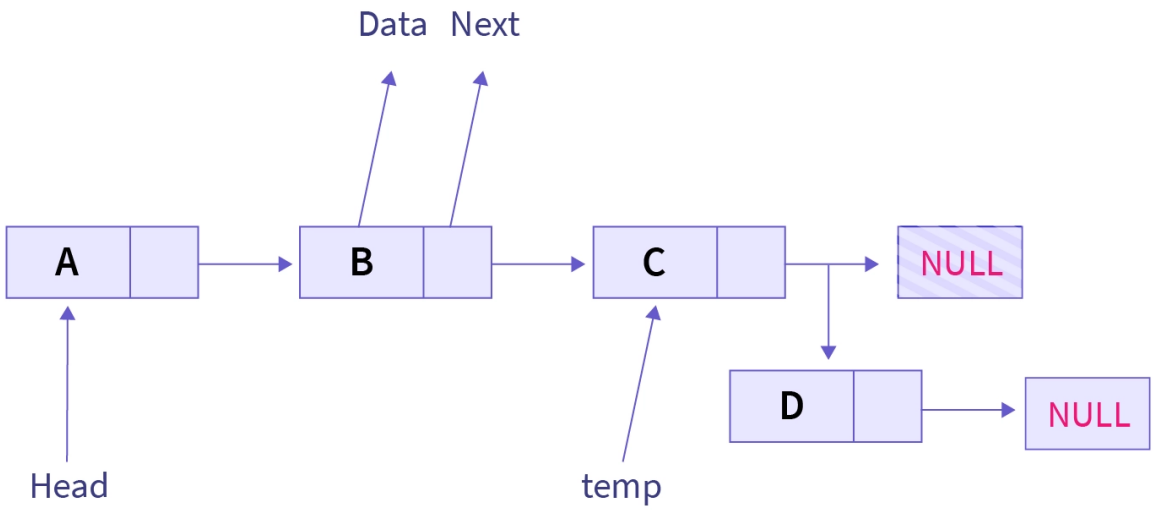
Basic LinkedList Functions & Operations

Many applications use LinkedList in computer science, let's discuss basic LinkedList functions.

- A node can be represented using structures.
- A node takes the form of a user-defined structure, a node contains two parts,i.e. to store data and to store the reference of the next node
- Basic LinkedList functions
  are create(), display(), insert_begin(), insert_end(), insert_pos(), delete_begin(), delete_end(), delete_pos()

create()

- This function is a foundation pillar for the entire linked list.
- Here, we create a temp node to scan the value.
- Then we check if LinkedList is empty or not, if LinkedList is empty then the temp node would be the head node.
- If LinkedList is not empty, then by using another node, we traverse till the end of LinkedList and add the temp node at the end of LinkedList.



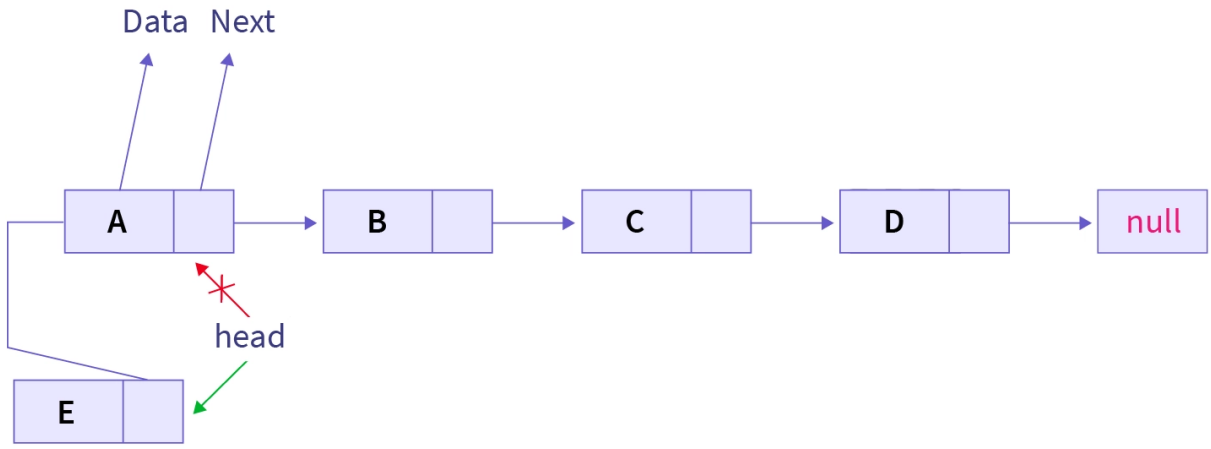
SCALER Topics

display()

- This function is used to display the entire LinkedList using a while loop
- We first check, if the head node is pointing to NULL or not, if the head node is pointing to NULL, then it indicates that LinkedList is empty, so we return
- If LinkedList is not empty, we assign head node to a temp node and we use this temp node to traverse over the LinkedList using a loop and print them

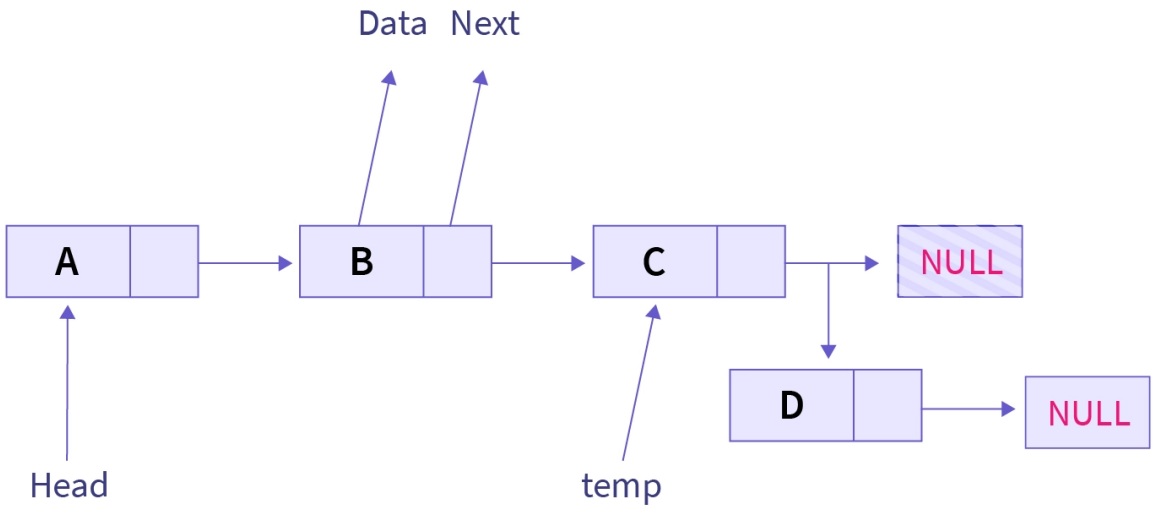
insert_begin()

- Initially, we create a temp node to scan the value then we check if LinkedList is empty or not
- If LinkedList is empty, then the newly created node would be treated as a head node
- If LinkedList is not empty, then we make the temp node point towards the current head node and the head node to point towards the newly created node


Data
Next
A
B
C
D
null
head
E

insert_end()

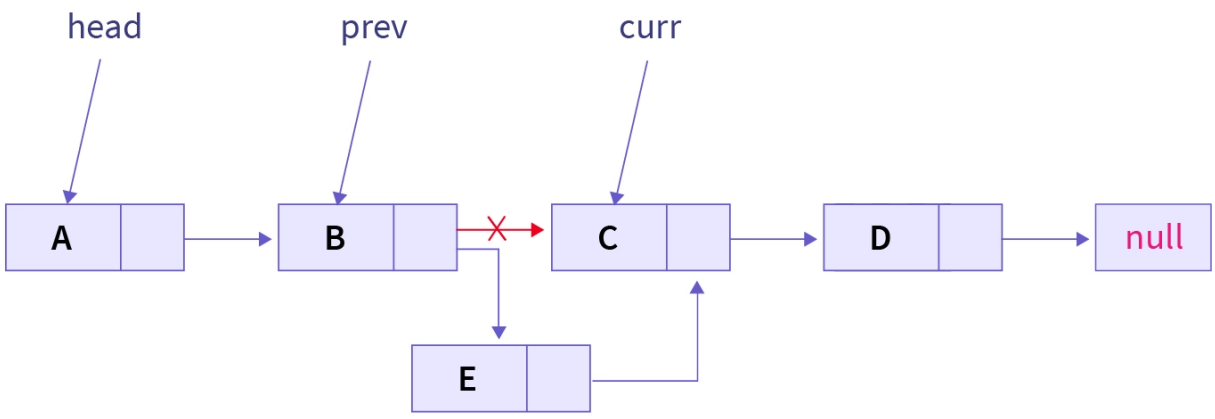
- Firstly, we create a temp node to scan the value then we check if LinkedList is empty or not
- If LinkedList is empty, then the newly created node would be inserted to LinkedList
- If LinkedList is not empty, then we create a new node say ptr, by using ptr we traverse till the end of LinkedList and insert the temp node at the end of LinkedList



SCALER Topics

insert_pos()

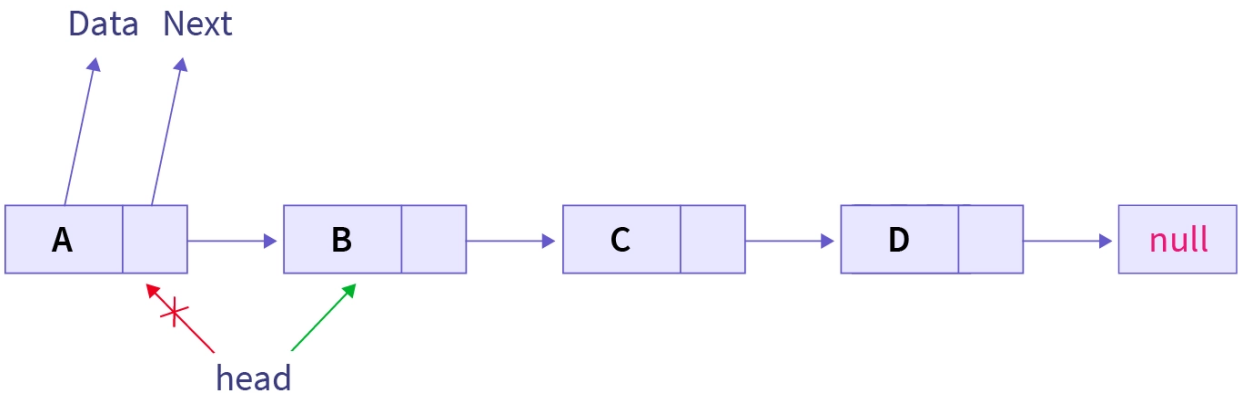
- Here, we create a temp node to scan the value then we check if LinkedList is empty or not
- If LinkedList empty, then we return
- If LinkedList is not empty, then we take input of node position from the user, if the input is greater than the length of LinkedList, then we return
- If the input is in the range of length of LinkedList then, let's assume we have four nodes A, B, C, D and we need to insert a node next to B, so, we just traverse till node C and make node B point to node E and node E to point to node C.



SCALER Topics

delete_begin()

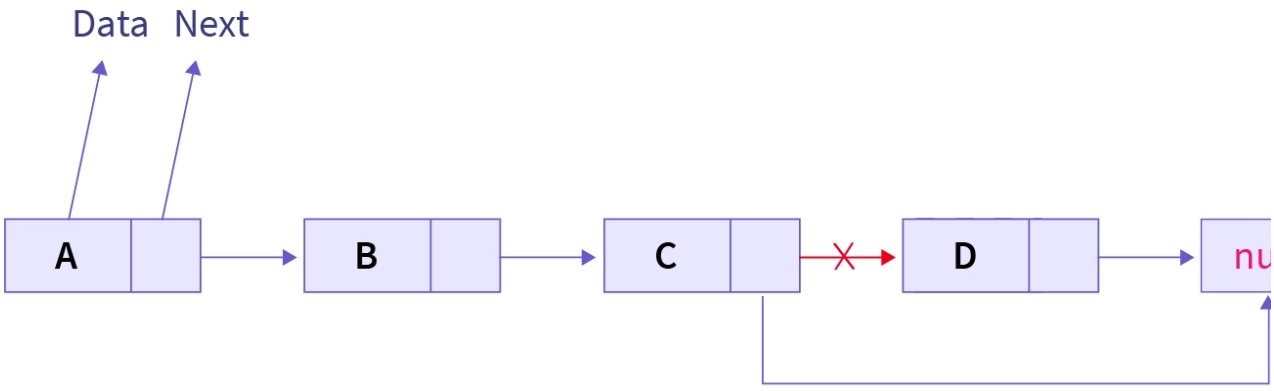
- This function checks if nodes are present in LinkedList or not, if nodes are not present then we return
- If nodes are present then we make ahead node to point towards the second node and store the address of the first node in a node say, temp
- By using the address stored in temp, we delete the first node from the memory



SCALER Topics

delete_end()

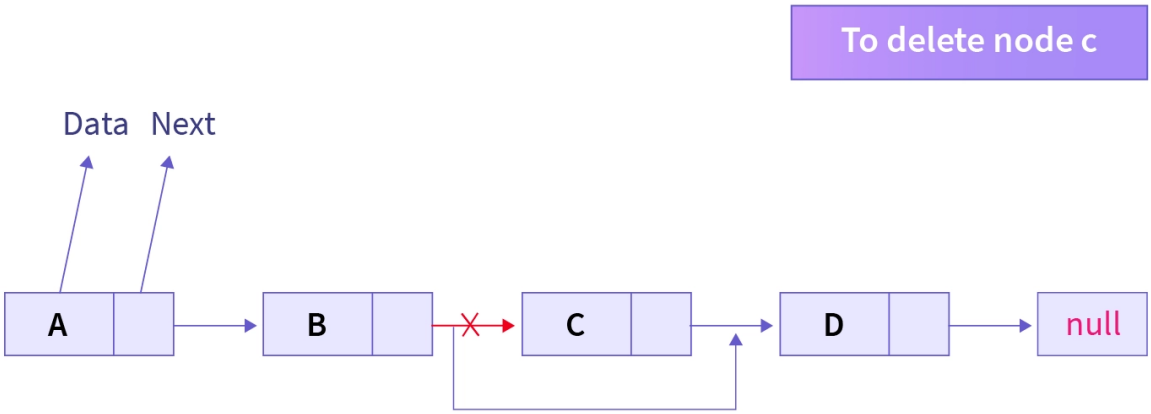
- This function checks if nodes are present in LinkedList or not, if nodes are not present in LinkedList, then we return
- If nodes are present in LinkedList, then we create a temp node and assign a head node value in it.
- By using this temp node, we traverse till last but one node of the LinkedList, and then we store the address present in the next field in a node say ptr.
- Now, we delete the ptr from memory, such that the last node is deleted from LinkedList



SCALER Topics

delete_pos()

- On invoking this function, we check if nodes are present in LinkedList or not, if nodes are not present then we return
- If nodes are present in LinkedList, as x,y,z and we need to delete node y
- To delete node y, we traverse till node x and make x to point towards node z, then we delete node y from memory



SCALER Topics

Constructing Linked List

Let's discuss multiple approaches to building a LinkedList

Naive Method for Creating LinkedList

The naive method for linked list implementation in C is to create individual nodes and link them later using the address of the nodes.

Let's create five nodes and link them later.

Implementation:

```
struct Node
{
        int data;
        struct Node* next;
};
int main()
{
        struct Node* node1 = (struct Node*)malloc(sizeof(struct Node));
        struct Node* node2 = (struct Node*)malloc(sizeof(struct Node));
        struct Node* node3 = (struct Node*)malloc(sizeof(struct Node));
        struct Node* node4 = (struct Node*)malloc(sizeof(struct Node));
        struct Node* node5 = (struct Node*)malloc(sizeof(struct Node));

        node1->data = 100;
        node2->data = 200;
        node3->data = 300;
        node4->data = 400;
        node5->data = 500;

        struct Node* head = node1;
        node1->next = node2;
        node2->next = node3;
        node3->next = node4;
        node4->next = node5;
        node5->next = NULL;

        struct Node* ptr = head;
        while(ptr!=NULL)
        {
                printf("%d ",ptr->data);
                ptr=ptr->next;
        }
}
```

- In the above code, initially we created a structure of type node
- Using this structure, we created five individual nodes and we also initialized data field for every node

- Then, using the address of the node we linked all the five nodes and made them a LinkedList
- This LinkedList is displayed by using a while loop

Single Line Approach for Creating LinkedList

- In the Naive approach, redundant code is present, so, let's discuss how to eliminate redundant code
- Here, nextnode is passed as an argument to the newNode(), this approach helps in eliminating redundant lines of code

Implementation:

```
struct Node
{
        int data;
        struct Node* next;
};
struct Node* newNode(int data, struct Node* nextNode)
{
        struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = data;
        temp->next = nextNode;
        return temp;
}
int main()
{
        struct Node* head = newNode(100, newNode(200, newNode(300,
newNode(400, newNode(500, NULL)))));
        struct Node* ptr = head;
        while(ptr!=NULL)
        {
                printf("%d ",ptr->data);
                ptr = ptr->next;
        }
}
```

- In the above code, initially we created a structure of type node
- We also created newNode function with data, node address as function parameters
- From the main function we are accessing newNode function with its parameters and we are creating a new node for a function call(newNode)
- And, we are returning the newly created address node to the main function, this address is again used to call newNode function
- Finally, by using the address of the head node we are printing the entire LinkedList

Generic Method for Creating LinkedList

- Naive Method and Single Line Method are suitable to understand the implementation of LinkedList

- But, these methods are not suitable to create n number of nodes
- If these methods are used to create n number of nodes, redundant code would be present
- In the below code, the array is traversed from right to left because the head node should be pointing to the first element in the array

Implementation:

```
struct Node
{
        int data;
        struct Node* next;
};
struct Node* newNode(int data,struct Node* nextNode)
{
        struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = data;
        temp->next = nextNode;

        return temp;
}
int main()
{
        int values[] = {100,200,300,400,500};
        int n = sizeof(values)/sizeof(values[0]);
        struct Node* head = NULL;
        struct Node* ptr;

        for(int i=n-1;i>=0;i--)
        {
                ptr    = newNode(values[i],ptr);
                head = ptr;
        }

        //printing LinkedList
        while(ptr->next != NULL)
        {
                printf("%d ",ptr->data);
                ptr = ptr->next;
        }
}
```

- In the above code, initially we created a structure of type node
- We also created newNode function with data, node address as function parameters
- In the main() we have created values array with integer values and stored the size of values array in 'n'
- From the main(), we are using a for loop to traverse over the array, for every element in the array we are calling the newNode function with its parameters
- For every newNode call, it creates a Node and returns the address of the newly created node to main()
- Finally, by using the address of the head node we are printing the entire LinkedList

Standard Solution for Creating LinkedList

- Here, we implement this method same as push() in Stack Data Structure
- We simply add every node to the next field of the head node

Implementation:

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

void push(struct Node** headRef, int data)
{
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));

    temp->data = data;
    temp->next = *headRef;

    *headRef = temp;
}

struct Node* createList(int keys[], int n)
{
    int i;
        struct Node* head = NULL;

    for (i = n - 1; i >= 0; i--) {
        push(&head, keys[i]);
    }

    return head;
}

int main(void)
{
    int values[] = {100,200,300,400,500};
    int n = sizeof(values)/sizeof(values[0]);

    struct Node* head = createList(values, n);

    struct Node* ptr = head;
    while (ptr)
    {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
}
```

- In the above code, initially we created a structure of type node
- We also created newNode function with data, node address as function parameters
- In main() we are calling createList() by passing values array and the size of the array

- In createList() we traverse the array from right to left where every value of the array is passed to push().
- In push(), a node is created for every call, and the address of the node is returned to createList()
- Finally, by using the address of the head node we are printing the entire LinkedList

Making Head Pointer Global

- As we already know that head node points to the first node in the LinkedList, here head nodes are made global
- As, the head node is made global, it can be accessed from any function

Implementation:

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

struct Node* head = NULL;

void push(int data)
{
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));

        temp->data = data;
    temp->next = head;

    head = temp;
}

int main()
{
    int i,n;
       int values[] = {100,200,300,400,500};
    n = sizeof(values)/sizeof(values[0]);

    for (i = n - 1; i >= 0; i--) {
        push(values[i]);
    }

    struct Node* ptr = head;
    while(ptr!=NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
}
```

- In the above code, initially we created a structure of type node

- In main() we created values array with Integer values, for every value in the array we are calling push function, where we are creating our nodes and linking them together
- As the head pointer is global, we are using it to print the entire LinkedList

Return Head From Push Function

- In this approach, the head node is not made global, the head node is passed as an argument to the push function
- Push function creates nodes and appends nodes to the LinkedList and then returns the head node to the main function

Implementation:

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

struct Node* push(struct Node* head, int data)
{
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));

        temp->data = data;
    temp->next = head;

    return temp;
}
int main()
{
        int i,n;
    int values[] = {100, 200, 300, 400, 500};
    n = sizeof(values)/sizeof(values[0]);

    struct Node* head;
    for (i = n - 1; i >= 0; i--) {
        head = push(head, values[i]);
    }

    struct Node* ptr = head;
    while (ptr)
    {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
}
```

- In the above code, initially we created a structure of type node
- In main() we created values array with Integer values, for every value in the array we are calling push function

- In push() a node is created for every value and the address of this node is returned to the main function
- By using the last address returned by push(), we print the entire LinkedList