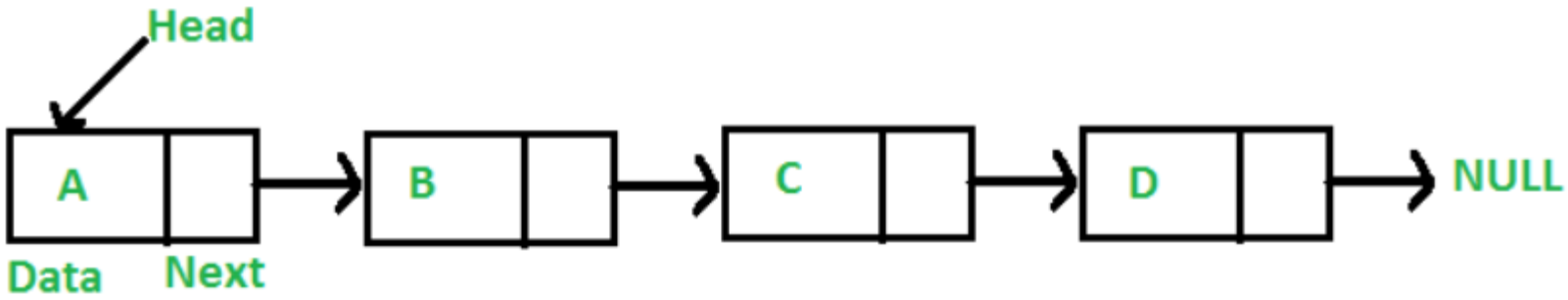
Linked List

Why linked lists?

- One disadvantage of using arrays to store data is that arrays are static structures.
- They cannot be easily extended or reduced to fit the data.
- Arrays are also expensive to maintain new insertions and deletions.
- A data structure called Linked Lists addresses some of the limitations of arrays.

Introduction

- A linked list is a linear data structure.
- The elements are not stored at contiguous memory locations.
- The elements in a linked list are linked using pointers .

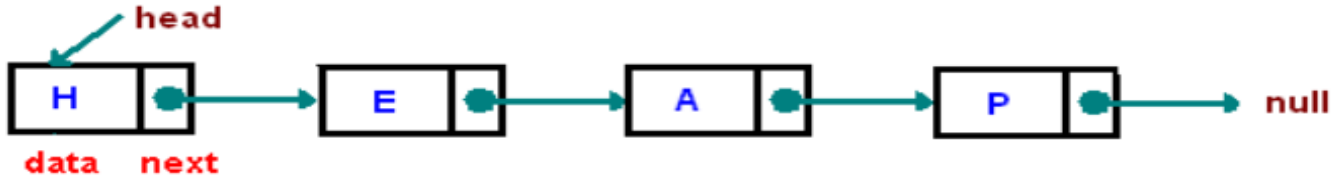


- A linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.
- The last node has a reference to null.

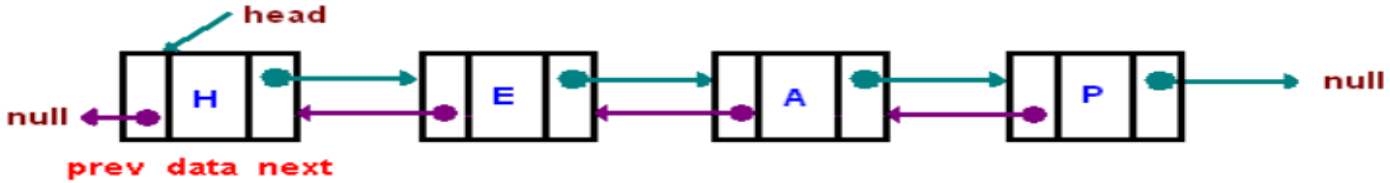
- A linked list is a dynamic data structure.
- The number of nodes in a list is not fixed and can grow and shrink on demand.
- One disadvantage of a linked list against an array is that it does not allow direct access to the individual elements.
- If one wants to access a particular item then we have to start at the head and follow the references until we get to that item.
- Another disadvantage is that a linked list uses more memory compared with an array.(4 extra for reference)

Types of linked lists

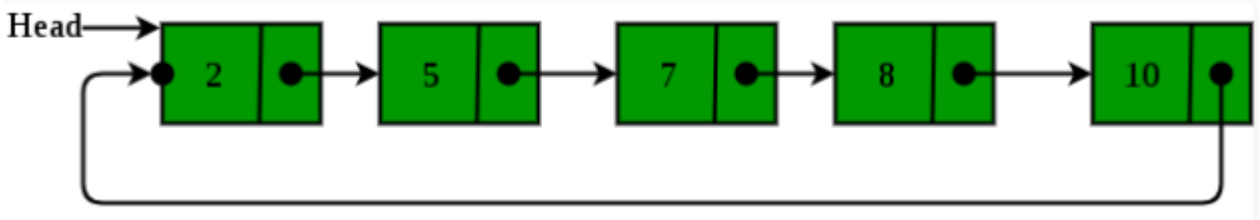
1. **Singly linked list**



2. **Doubly linked list**



3. **Circular linked list**



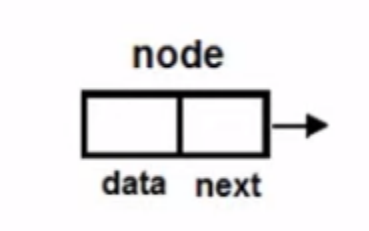
Singly linked list

- Basic operations

    1. Insertion
    2. Deletion
    3. Display
    4. Search

Create a node

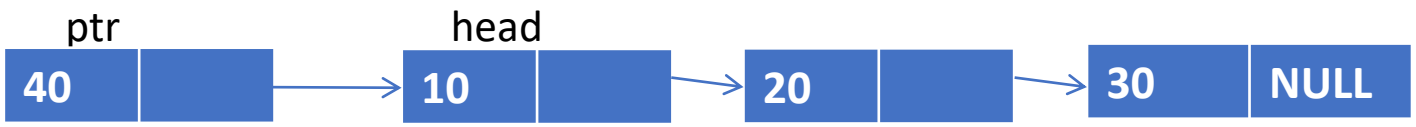
```
struct node
{
        int data;
        struct node *next;
};
```



Insertion operation

1. Insert at beginning of list
   1. The next pointer of the node created will be made to point the head of the current list.
   2. The new node is marked as the head node.

Call to malloc
newnode->data = item
newnode → next = head;
head = newnode



2. Middle of a list
   1. The next pointer of the previous node will point to the new node.
   2. The next pointer of the new node will point to the next node.

```
for(i=0 till position)
{p=p->next;}
newnode ->next = temp ->next;
temp ->next = newnode;
```

3. Insert at end

1. The next pointer of the last node is made to point at the new node.
2. The next pointer of the new node is made to point to NULL.

```
while (p->next!=NULL)
{p=p->next;}
P->next = newnode;
newnode->next = NULL
```

Delete node

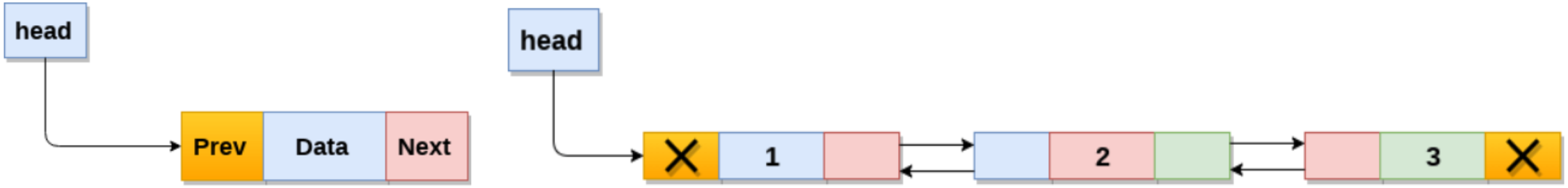
1. Delete at beginning
    1. Make the next node as the new head.
    2. Delete link and then free memory of the first node.

2. Delete at the end
    1. Make the second last node to point to null.
    2. Delete link and free memory of the last node.

3. Delete intermediate node
    1. Point the next pointer of the previous node to point to the next node
    2. Delete link and free the memory of the deleted node.

Doubly linked list

- Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node.
- A node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer)



Structure creation for a doubly linked list

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

Circular linked list

- In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list.
- We can have circular singly linked list as well as circular doubly linked list.
- The circular singly liked list has no beginning and no ending.
- There is no null value present in the next part of any of the nodes.