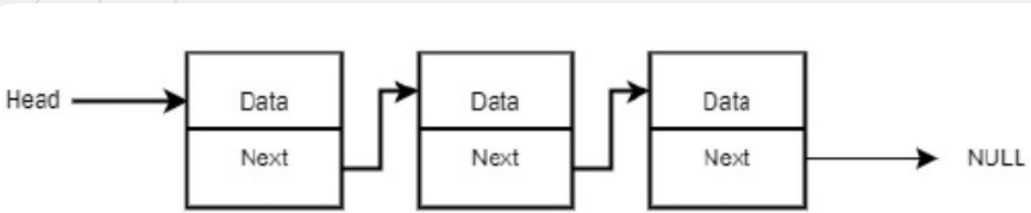

Linked List

Definition:

A linked list is a collection of **“nodes”** connected together via links. These nodes consist of the **data** to be stored and **a pointer to the address of the next node** within the linked list. In linked lists, there is no defined size. Any amount of data can be stored in it and can be deleted from it.

Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.


Head
Data
Next
Data
Next
Data
Next
NULL

1

Important Points

Singly Linked List

Important Points:

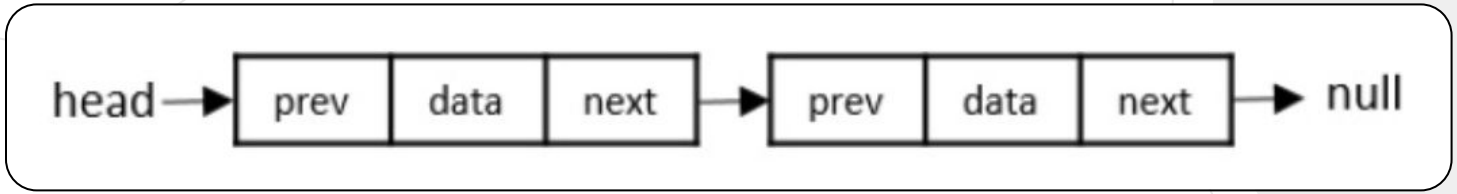
- Linked List contains a link element called first (**head**).
- Each link carries a **data field(s)** and a link field called **next**.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.
- Optional last (**tail**).

Types of Linked Lists:

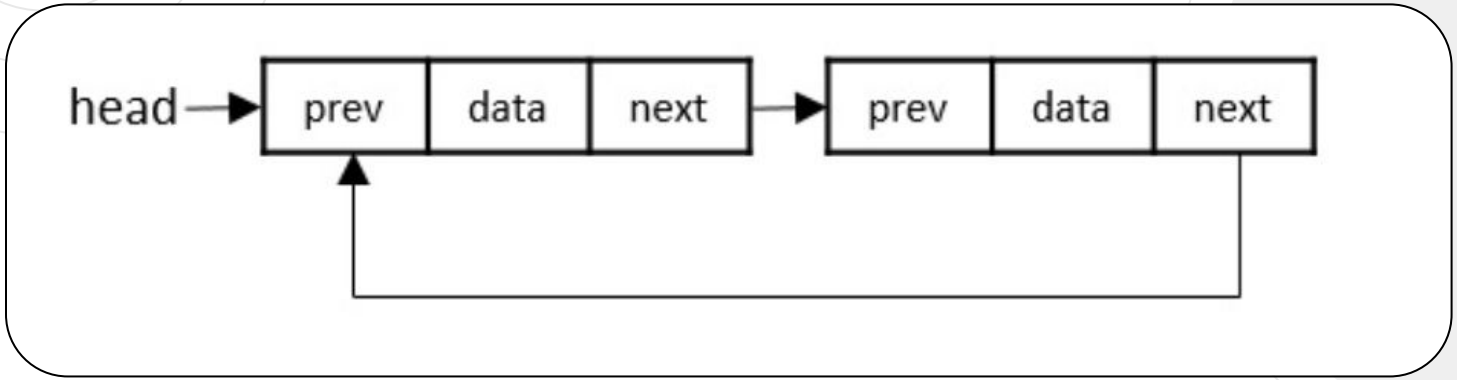
1. Singly Linked Lists
2. Doubly Linked Lists
3. Circular Linked Lists

Singly Linked List

Doubly Linked List



Circular Linked List



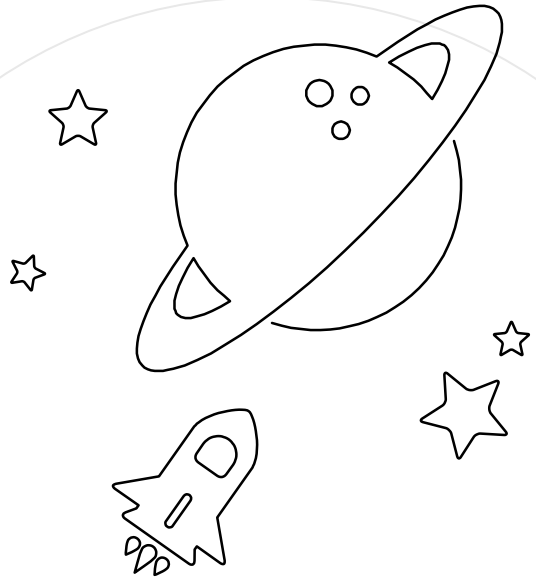
1

Basic Operations

Singly Linked List

Basic Operations:

1. **Insertion** - Inserts an element.
2. **Deletion** - Deletes an element.
3. **Print** - Prints the complete list.
4. **Search** - Searches an element in the give linked list.



Insertion

Beginning | Between | End

Beginning

```
/* Inserts a new Node at front of the list. */
public void addStart(int new_data)
{
    /* 1 & 2: Allocate the Node & Put in the data*/
    Node new_node = new Node(new_data);

    /* 3. Make next of new Node as head */
    new_node.next = head;

    /* 4. Move the head to point to new Node */
    head = new_node;
}
```

Time Complexity: O(1)

Space Complexity: O(1)

Between

1. Check if the given node exists or not.
    a. If it do not exists,
        i. terminate the process.
    b. If the given node exists,
        i. Make the element to be inserted as a new node
        ii. Change the next pointer of given node to the new node
        iii. Now shift the original next pointer of given node to the next pointer of new node

Between

```
/* Inserts a new node after the given prev_node. */
public void insertAfter(Node prev_node, int new_data)
{
        /* 1. Check if the given Node is null */
        if (prev_node == null) return;

        /* 2 & 3: Allocate the Node & Put in the data*/
        Node new_node = new Node(new_data);

        /* 4. Make next of new Node as next of prev_node */
        new_node.next = prev_node.next;

        /* 5. make next of prev_node as new_node */
        prev_node.next = new_node;
}
```

Time Complexity: O(1)

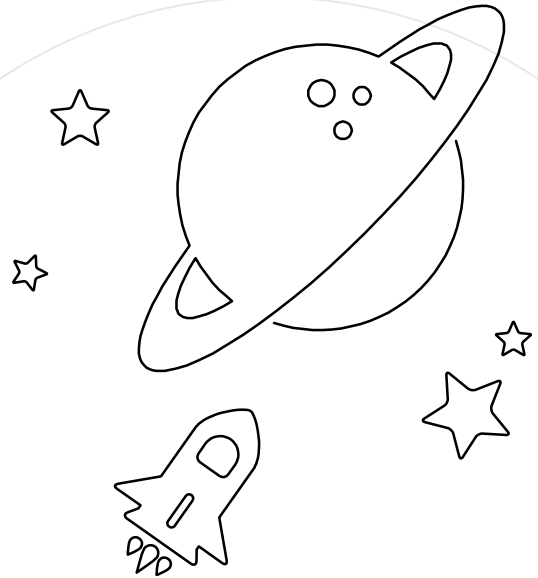
Space Complexity: O(1)

End

```
public void append(int new_data)
{
        /* 1. Allocate the Node &
        2. Put in the data */
        Node new_node = new Node(new_data);

        /* 3. If the Linked List is empty, then make the new node as head */
        if (head == null) {
                head = new Node(new_data);
                return;
        }

        /* 4. This new node is going to be the last node, so make next of it as null */
        new_node.next = null;

        /* 5. Else traverse till the last node */
        Node last = head;
        while (last.next != null) last = last.next;

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

Time Complexity: O(N)

Space Complexity: O(1)



Deletion

Beginning | Between | End

Beginning

```
if (head == null)
{
        Console.WriteLine("The list is empty. Nothing to delete.");
        return;
}

head = head.Next;
```

Time Complexity: $O(1)$

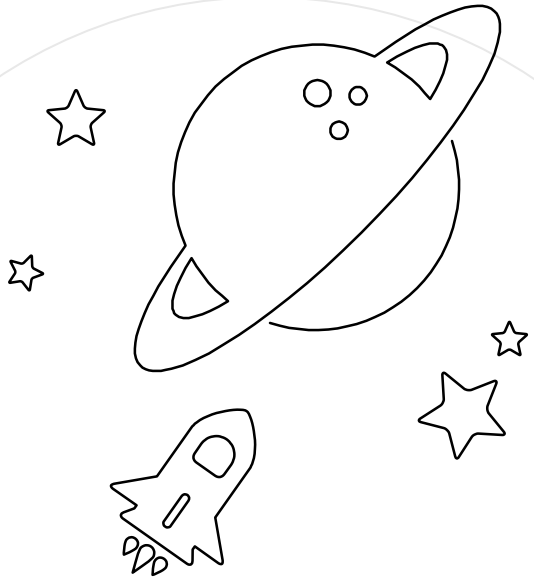
Space Complexity: $O(1)$

End

```
if (head == null)
{
        Console.WriteLine("The list is empty. Nothing to delete.");
        return;
}

if (head.Next == null)
{
        // If there's only one element in the list, set head to null.
        head = null;
        return;
}

Node current = head;

while (current.Next.Next!= null)
{
        current = current.Next;
}
        current.Next = null;
```

Time Complexity: O(N)

Space Complexity: O(1)



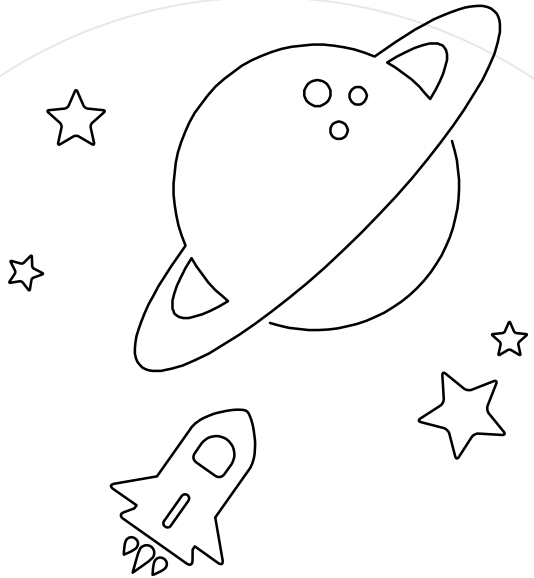
Print

Print

```
public static void printList(Node head)
{
        while (head != null) {
                if (head.next == null) {
                        Console.WriteLine("[" + head.data + "] " + "[" + head + "]->" + "(nil)");
                }

                else {
                        Console.WriteLine("[" + head.data + "] " + "[" + head + "]->" + head.next);
                }

                head = head.next;
        }
        Console.WriteLine("\n");
}
```

Time Complexity: O(N)

Space Complexity: O(1)



Search

Search

```
public bool Search(int target)

{
        Node current = head;

        while (current != null)
        {
                if (current.data == target)
                {
                        return true; // Element found
                }
                current = current.Next;
        }

          return false; // Element not found
}
```

Time Complexity: O(N)

Space Complexity: O(1)

Thanks!

Any questions?