# Linkedlist

## Introduction

A **linked list** is a fundamental data structure in computer science. It consists of nodes, where each node contains data and a reference (link) to the next node in the sequence. Unlike arrays, linked lists allow for efficient insertion or removal of elements from any position in the list, as the nodes are not stored contiguously in memory[13]. Here are the key points:

- **Definition**: A linked list is a linear data structure consisting of a group of nodes where each node points to the next node through a pointer.
- **Node Structure**: Each node contains two components:
  - **Data**: The actual value or payload stored in the node.
  - **Next Pointer**: A reference to the next node in the list.
- **Advantages**:
  - Dynamic Memory Allocation: Linked lists allocate memory dynamically, allowing efficient use of system memory.
  - Efficient Insertion/Deletion: Adding or removing elements is faster compared to arrays.
- **Types of Linked Lists**:
  - **Singly Linked List**: Each node points to the next node.
  - **Doubly Linked List**: Nodes have both forward and backward pointers.
  - **Circular Linked List**: The last node points back to the first node.
- **Applications**:
  - Implementing stacks and queues using linked lists.
  - Handling collisions in hash tables.
  - Representing graphs.
  - Dynamic memory allocation.

Remember, linked lists are like a chain of interconnected nodes, allowing flexibility and efficient data management! [12]

## Implementation

### Python Linked List Implementation

In Python, we can create a simple linked list using classes. Here's an example:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
```

```
    def insert_at_beginning(self, value):
        new_node = Node(value)
        new_node.next = self.head
        self.head = new_node

    def insert_at_end(self, value):
        new_node = Node(value)
        if not self.head:
            self.head = new_node
            return
        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node

    def insert_at_position(self, value, position):
        if position < 1:
            print("Position should be >= 1.")
            return
        new_node = Node(value)
        if position == 1:
            new_node.next = self.head
            self.head = new_node
            return
        temp = self.head
        for _ in range(position - 2):
            if temp:
                temp = temp.next
            else:
                print("Invalid position.")
                return
        if not temp:
            print("Invalid position.")
            return
        new_node.next = temp.next
        temp.next = new_node

    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")

# Example usage:
```

```
my_list = LinkedList()
my_list.insert_at_beginning(10)
my_list.insert_at_end(20)
my_list.insert_at_position(15, 2)
my_list.display()
```

### C++ Linked List Implementation (using `std::list`)

In C++, we can use the `std::list` container from the Standard Template Library (STL) to create a doubly linked list. Here's an example:

```
#include <iostream>
#include <list>

int main() {
    std::list<int> my_list = {12, 45, 8, 6};

    // Display the elements
    for (const auto& item : my_list) {
        std::cout << item << " ";
    }
    std::cout << std::endl;

    // Other basic operations:
    std::cout << "Front element: " << my_list.front() << std::endl;
    std::cout << "Back element: " << my_list.back() << std::endl;

    my_list.pop_front();
    std::cout << "After popping front: ";
    for (const auto& item : my_list) {
        std::cout << item << " ";
    }
    std::cout << std::endl;

    my_list.pop_back();
    std::cout << "After popping back: ";
    for (const auto& item : my_list) {
        std::cout << item << " ";
    }
    std::cout << std::endl;

    return 0;
}
```