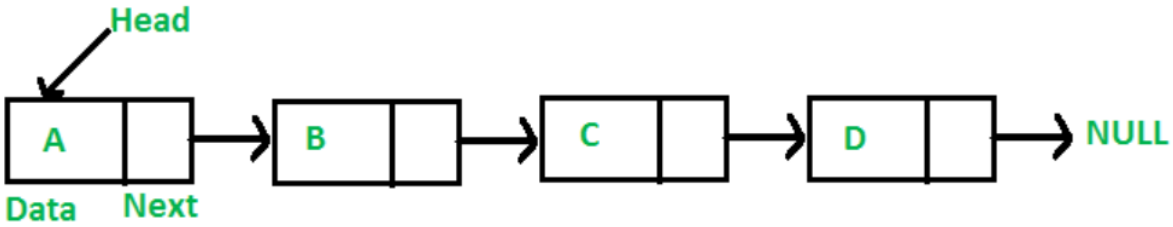
<Linked List>



연속적인 메모리 위치에 저장되지 않는 선형 데이터 구조

(포인터를 사용해서 연결된다)

각 노드는 **데이터 필드**와 **다음 노드에 대한 참조**를 포함하는 노드로 구성

왜 Linked List 를 사용하나?

배열은 비슷한 유형의 선형 데이터를 저장하는데 사용할 수 있지만 제한 사항이 있음

1) 배열의 크기가 고정되어 있어 미리 요소의 수에 대해 할당을 받아야 함
2) 새로운 요소를 삽입하는 것은 비용이 많이 듬 (공간을 만들고, 기존 요소 전부 이동)

장점

1) 동적 크기
2) 삽입/삭제 용이

단점

1) 임의로 액세스를 허용할 수 없음. 즉, 첫 번째 노드부터 순차적으로 요소에 액세스 해야함 (이진 검색 수행 불가능)
2) 포인터의 여분의 메모리 공간이 목록의 각 요소에 필요

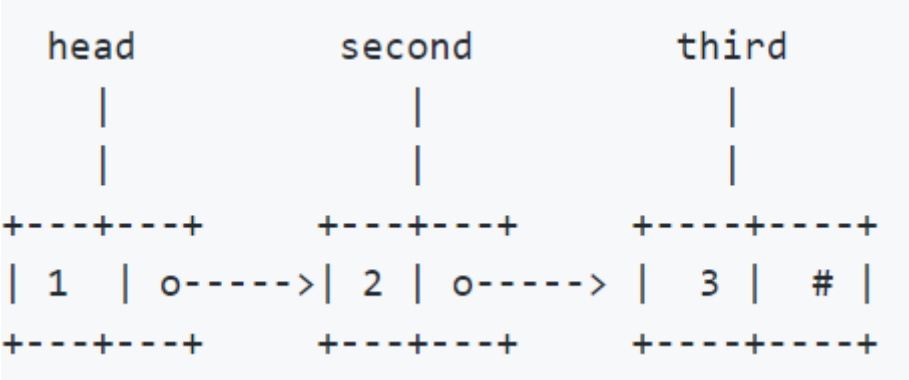
노드 구현은 아래와 같이 데이터와 다음 노드에 대한 참조로 나타낼 수 있다

```go
type Node struct {
    data int
    next *Node
}
```

1. Single Linked List

https://github.com/limes22/algorithm/blob/main/go/DataStructure/LinkedList/linked_list.go

노드 3 개를 잇는 코드를 만들어본다.



2. 노드 추가

https://github.com/limes22/algorithm/blob/main/go/DataStructure/LinkedList/linked_list_push.go

- 앞쪽에 노드 추가

```go
func (m *Main) Push(new_data int) {
    new_node := new(Node)
    new_node.data = new_data
    new_node.next = m.head
    m.head = new_node
}
```

- 특정 노드 다음에 추가

```
func (m *Main) InsertAfter(prev_node *Node, new_data int) {
    if prev_node == nil {
        fmt.Println( a...: "The given Previous node canot be null")
        return
    }
    new_node := new(Node)
    new_node.data = new_data
    new_node.next = prev_node.next
    prev_node.next = new_node
}
```

- 끝쪽에 노드 추가

```
func (m *Main) Append(new_data int) {
    new_node := new(Node)
    new_node.data = new_data
    if m.head == nil {
        m.head = new(Node)
        m.head.data = new_data
        return
    }
    new_node.next = nil
    last := m.head
    if last.next != nil {
        last = last.next
        last.next = new_node
        return
    }
}
```

<Reference>

https://github.com/gyoogle/tech-interview-for-developer/blob/master/Computer%20Science/Data%20Structure/Linked%20List.md