SWIFT
Data Structure - Linked List

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Linked List

링크드 리스트(Linked List)는 순차적으로 모인 데이터의 모음으로서 다음 차례의 노드 주소를 가지고 있는 형태를 가집니다.

가지고 있는 노드의 주소 형태에 따라서 아래의 두 가지 유형을 나타낼 수 있습니다.

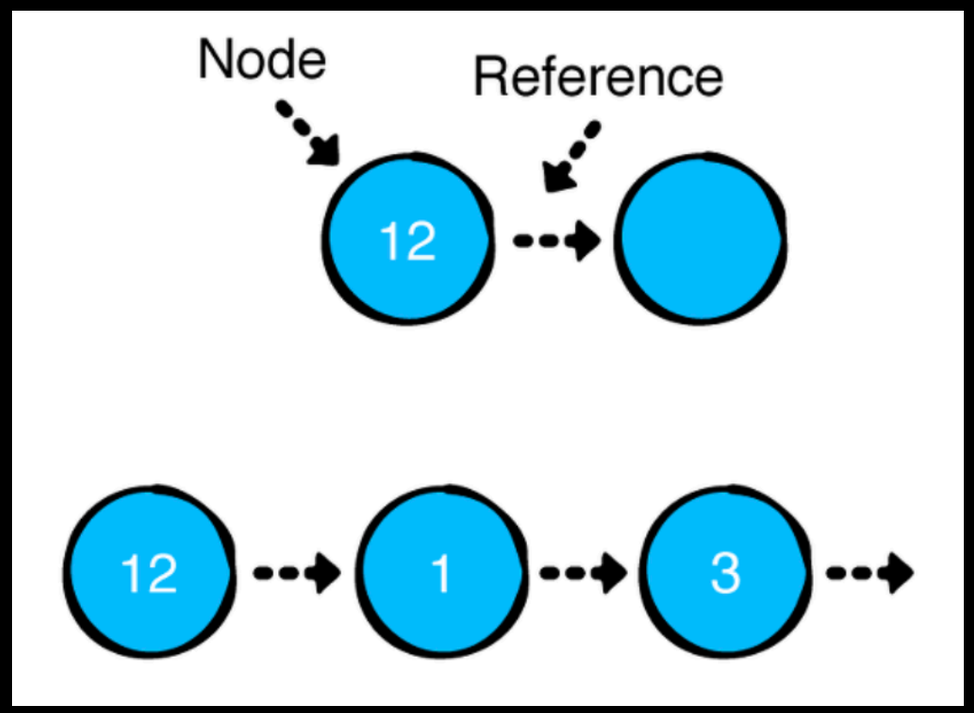
Singly Linked List :
다음 노드(Next Node)의 주소만 가지는 리스트

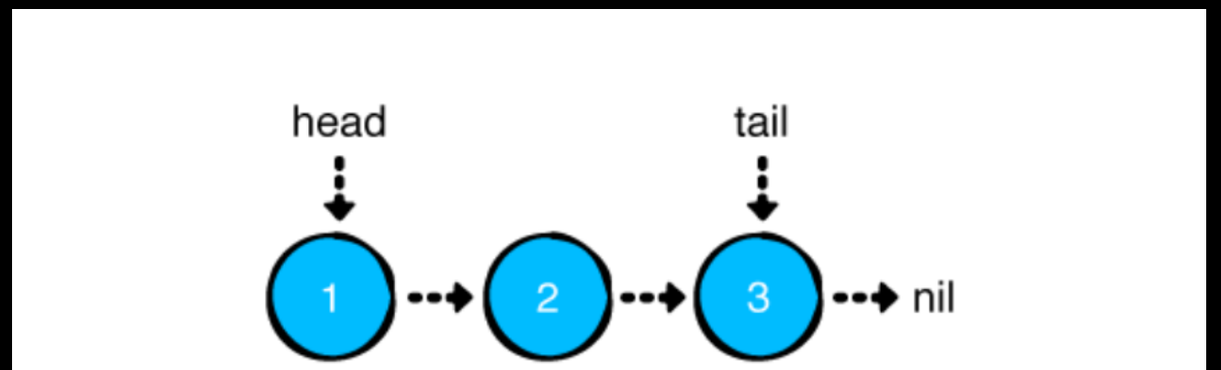
Double Linked List :
다음 노드 및 이전 노드(Previous Node)의 주소를 가지는 리스트

Linked List를 사용하기 위해서는 **시작과 끝**의 주소를 알아야 하는데 이를 **head**와 **tail**이라 부릅니다.

Concept

Linked List의 기본 동작 흐름은 아래와 같습니다.





The head and tail of the list

Features

Linked List의 특징을 살펴보면 아래와 같습니다.

- 데이터를 순차적으로 동적 저장합니다.
- 데이터 중복 저장을 허용합니다.
- 총 길이의 제한이 없습니다.
- 특정 노드의 주소를 모르면 직접 접근이 불가합니다.

Implementation

Swift를 활용하여 Linked List 를 구현해보겠습니다.
우선 필요한 메소드는 아래와 같습니다.

- init : 리스트를 초기화하는 함수
- insert : 데이터 입력(마지막 혹은 특정 노드 위치)
- remove : 특정 노드 삭제
- removeLast : 마지막 데이터 삭제
- removeAll : 모든 데이터 삭제
- count : 현재 리스트의 크기를 반환
- isEmpty : 현재 리스트의 크기가 비어있는지 체크

Implementation

가장 데이터의 기본이 되는 Node 클래스 입니다.

해당 Node 클래스는 모든 데이터 형식을 받을 수 있도록
Generic 형태로 구현이 되어 있습니다.

```
class LinkedListNode<T> {
    var value: T
    var next: LinkedListNode?
    weak var previous: LinkedListNode?

    public init(value: T) {
        self.value = value
    }
}
```

Implementation

```
class LinkedList<T> {
    typealias Node = LinkedListNode<T>

    private var head: Node?
    private var tail: Node?

    public init() {
        head = nil
        tail = nil
    }

    public var isEmpty: Bool {
        return head == nil
    }

    public var first: Node? {
        return head
    }

    public var last: Node? {
        return tail
    }

    ....
```

Implementation

```
public func node(at index: Int) -> Node? {
    if index >= 0 {
      var node = head
      var i = index
      while node != nil {
        if i == 0 { return node }
        i -= 1
        node = node!.next
      }
    }
    return nil
}

public func insert(_ value: T) {
    let newNode = Node(value: value)

    if let tailNode = tail {
      newNode.previous = tailNode
      tailNode.next = newNode
    } else {
      head = newNode
    }

    tail = newNode
}

....
```

Implementation

```
public func insert(_ node: Node, at index: Int) {
    if index == 0,
        tail == nil {
        head = node
        tail = node
    } else {
        guard let nodeAtIndex = self.node(at: index) else {
            print("Index out of bounds.")
            return
        }

        if nodeAtIndex.previous == nil {
            head = node
        }

        node.previous = nodeAtIndex.previous
        nodeAtIndex.previous?.next = node

        node.next = nodeAtIndex
        nodeAtIndex.previous = node
    }
}

....
```

Implementation

```
public func removeAll() {
    head = nil
    tail = nil
}

public func removeLast() -> T {
    return remove(node: last!)
}

public func remove(node: Node) -> T {
    let prev = node.previous
    let next = node.next

    if let prev = prev {
        prev.next = next
    } else {
        head = next
    }

    next?.previous = prev

    node.previous = nil
    node.next = nil

    return node.value
}

....
```

Implementation

```
public func count() -> Int {
    guard var node = head else {
        return 0
    }

    var count = 1
    while let next = node.next {
        node = next
        count += 1
    }

    return count
}

....
```

Implementation

```
public var toString : String {
    var s = "["
    var node = head
    while node != nil {
        s += "\(node!.value)"
        node = node!.next
        if node != nil { s += ", " }
    }
    return s + "]"
}
```

```
    struct LinkedListIterator : IteratorProtocol {
        let linkedList: LinkedList
        var current: Node?

        init(_ linkedList: LinkedList) {
            self.linkedList = linkedList
            self.current = linkedList.head
        }

        mutating func next() -> Node? {
            guard let thisCurrent = current else { return nil }
            current = thisCurrent.next
            return thisCurrent
        }
    }
}
```

Implementation

```swift
extension LinkedList : Sequence {
    func makeIterator() -> LinkedListIterator {
        return LinkedListIterator(self)
    }
}

// 사용 예시

let list:LinkedList<Int> = LinkedList<Int>()

list.insert(1)
list.insert(2)
list.insert(3)
list.insert(4)
list.insert(5)

// 현재 리스트 카운트 : 5
print(list.count())

for node in list {
    print(node.value)
    // 1
    // 2
    // 3
    // 4
    // 5
}
```

References

[1] 스위프트 : 연결리스트 (1 / 3) : #LinkedList : #DataStructrue : #자료구조: #swift : https://the-brain-of-sic2.tistory.com/14

[2] 스위프트 : 연결리스트 (2 / 3) : #LinkedList : #값 추가하기, push, append : #값 삽입하기,insert: #swift : https://the-brain-of-sic2.tistory.com/15

[3] [Swift 자료구조 ch08] Linked List : https://kor45cw.tistory.com/244

[4] Swift로 자료구조, 알고리즘 공부하기 (4) - Linked List : https://kor45cw.tistory.com/4

[5] Data Structure) 링크드 리스트(Linked List) in Swift : https://atelier-chez-moi.tistory.com/90

References

[6] Data Structure (자료구조) : https://opentutorials.org/module/1335

[7] Linked List : https://woongsios.tistory.com/49

[8] Data Structure 데이터 구조 : https://m.blog.naver.com/PostView.nhn?blogId=jdub7138&logNo=220957839382&proxyReferer=https%3A%2F%2Fwww.google.com%2F

[9] C <18>. 연결리스트 (Linked list) - 자료구조(1) : https://blog.yagom.net/249/

[10] [Data Structure] 자료구조 - 연결 리스트(Linked List) - 단순 연결 리스트(Singly / Linear Linked List) : https://palpit.tistory.com/141

Thank you!