```swift
import UIKit

/// 剑指 Offer 06. 从尾到头打印链表
/// 输入一个链表的头节点，从尾到头反过来返回每个节点的值（用数组返回）。
/// 输入：head = [1,3,2]
/// 输出：[2,3,1]
public class ListNode {
    public var val: Int
    public var next: ListNode?

    public init() {
        self.val = 0
        self.next = nil
    }
    public init(_ val: Int) {
        self.val = val
        self.next = nil
    }
    public init(_ val: Int, _ next: ListNode?) {
        self.val = val
        self.next = next
    }
}


func generateNode(_ list: [Int]) -> ListNode? {
    if let first = list.first {
        var listNode = ListNode(first)

        var temp_Node_2 = listNode

        for val in list.dropFirst() {
            let temp_Node = ListNode(val)
            temp_Node_2.next = temp_Node
            if let next = temp_Node_2.next {
                temp_Node_2 = next
            }
        }
        return listNode
    } else {
        return nil
    }
}

class Solution {
    /// 执行用时：16 ms, 在所有 Swift 提交中击败了 65.74% 的用户
    /// 内存消耗：15.5 MB, 在所有 Swift 提交中击败了 37.96% 的用户
    func reversePrintSolution1(_ head: ListNode?) -> [Int] {
        var result: [Int] = []
        var startNode = head
        while startNode != nil {
            if let val = startNode?.val {
                result.insert(val, at: 0)
            }
            startNode = startNode?.next
        }

        return result
    }
    /// 执行用时：16 ms, 在所有 Swift 提交中击败了 65.74% 的用户
    /// 内存消耗：15.5 MB, 在所有 Swift 提交中击败了 37.96% 的用户
    func reversePrint(_ head: ListNode?) -> [Int] {
        var result: [Int] = []

        var startNode = head
        while startNode != nil {
            if let val = startNode?.val {
                result.append(val)
            }
            startNode = startNode?.next
        }

        return result.reversed()
    }
}

let solution = Solution()
let head1 = generateNode([1,3,2])

print(solution.reversePrint(head1))
```