CHAPTER 04

단순 연결 리스트

학습목표

- 단순 연결 리스트의 개념을 파악한다.
- 단순 연결 리스트와 선형 리스트의 차이를 이해한다.
- 단순 연결 리스트의 데이터 삽입/삭제 원리를 이해한다.
- 파이썬으로 단순 연결 리스트를 조작하는 코드를 작성한다.

- **단순 연결 리스트란?**
  - 방문할 맛집을 지도에 순서대로 연결한 것처럼, 떨어진 곳에 위치한 데이터를 화살표로 연결한 것

방문 순서	1	2	3	4	5
지역	속초	대구	대전	부산	목포
이름	만석 닭강정	제일콩국	광천식당	황산냉밀면	하당먹거리

Section 01 단순 연결 리스트의 기본

- **단순 연결 리스트의 개념**
  - 노드들이 물리적으로 떨어진 곳에 위치
  - 각 노드의 번지도 순차적이지 않음
  - 화살표로 표시된 연결(링크, Link)을 따라가면 선형 리스트 순서와 같음

그림 4-1 선형 리스트

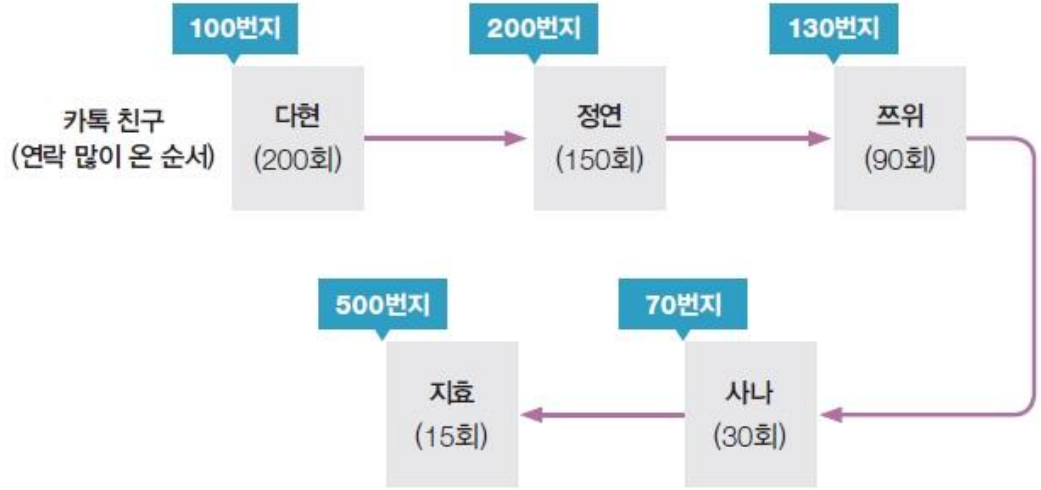


그림 4-2 단순 연결 리스트

Section 01 단순 연결 리스트의 기본

- 데이터를 삽입/삭제할 때
  - 선형 리스트는 많은 작업이 필요(오버헤드 발생)
  - 단순 연결 리스트는 해당 노드의 앞뒤 링크만 수정하면 되므로 오버헤드가 거의 발생하지 않음

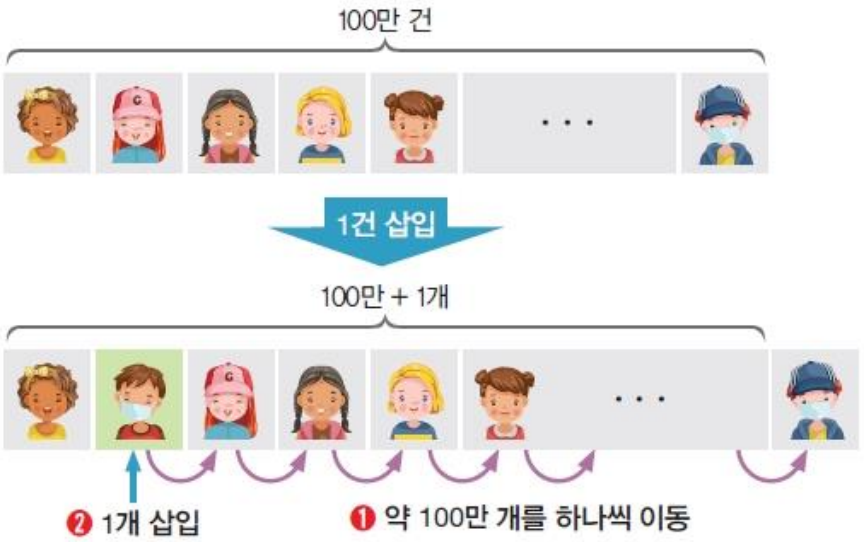


그림 4-3 선형 리스트에서 데이터 삽입 : 오버헤드 발생

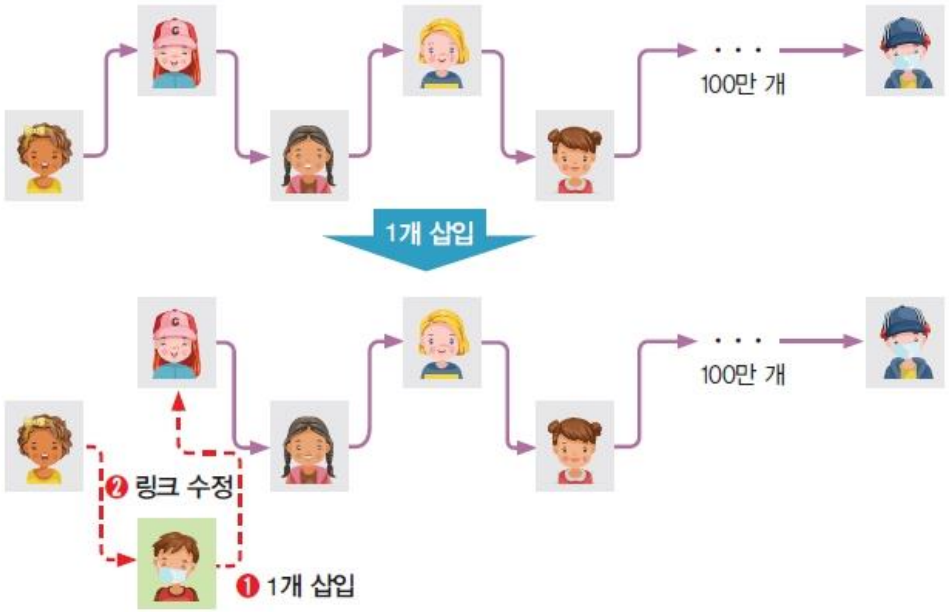


그림 4-4 단순 연결 리스트에서 데이터 삽입 : 오버헤드 없음

단순 연결 리스트의 원리

노드 구조

- 단순 연결 리스트는 다음 데이터를 가리키는 링크가 더 필요
- 노드(Node)는 데이터와 링크로 구성된 항목

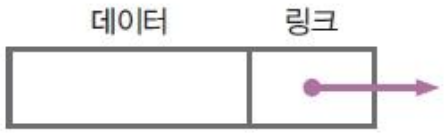


그림 4-5 노드 구조

↓ [그림 4-2]의 예를 노드로 표현

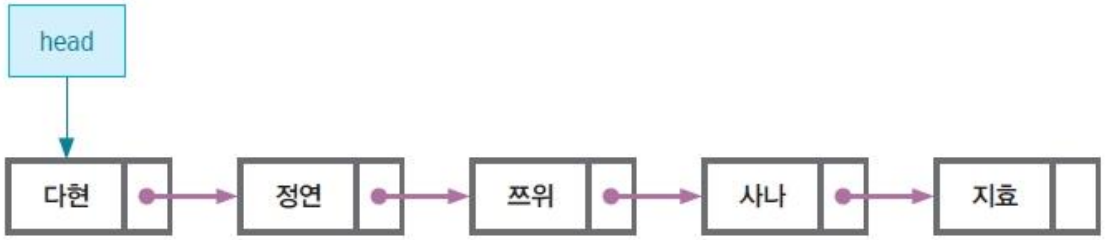


그림 4-6 노드로 구성된 단순 연결 리스트

Section 01 단순 연결 리스트의 기본

- 노드(데이터) 삽입

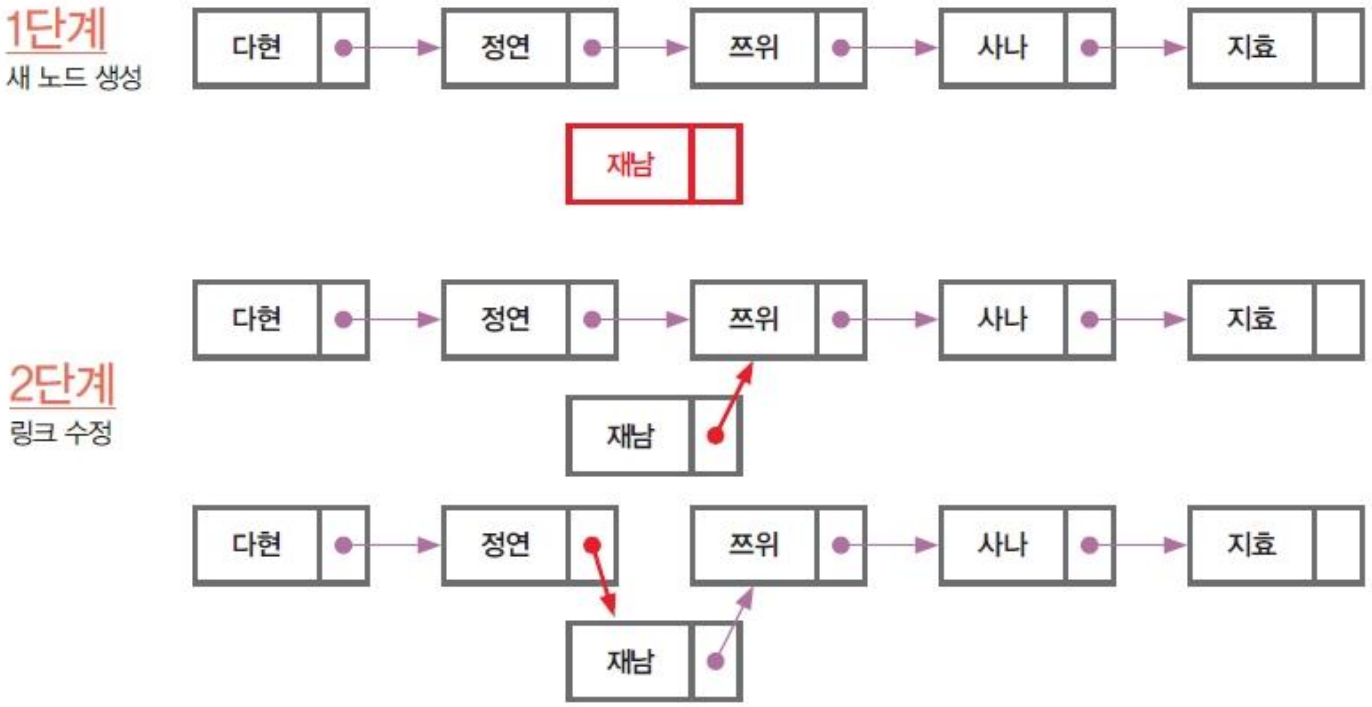


그림 4-7 단순 연결 리스트에서 데이터를 삽입하는 과정 예

- 노드(데이터) 삭제

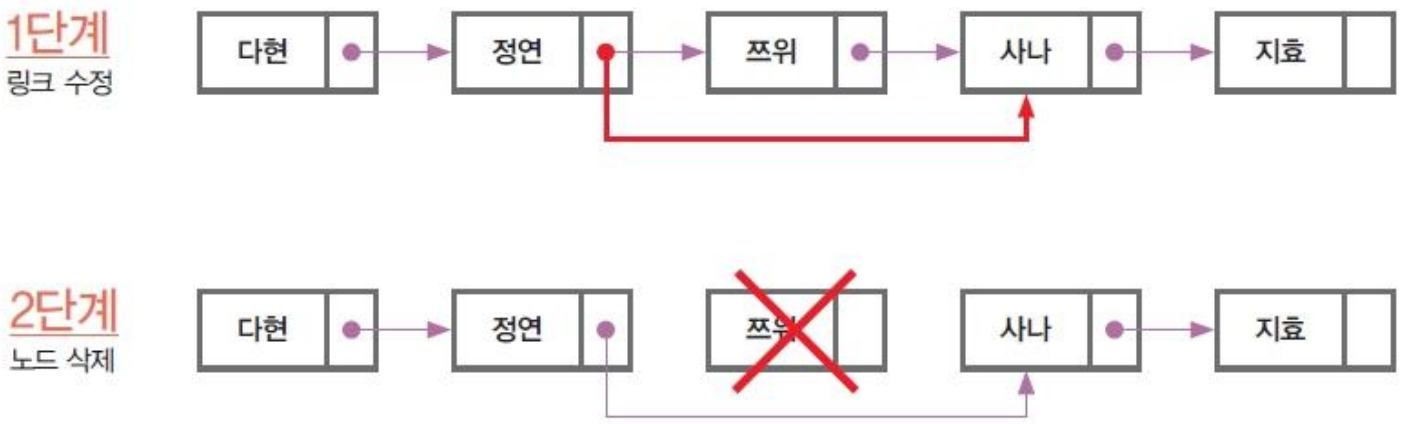


그림 4-8 단순 연결 리스트에서 데이터를 삭제하는 과정 예

■ 노드 생성과 연결

▪ 노드 생성

- 클래스라는 문법을 사용하여 Node 데이터형 정의

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

❶ Node라는 데이터형을 만드는 것

❷ 데이터형을 생성할 때 자동으로 실행되는 부분

❸ 데이터와 링크가 저장되는 부분

- 첫 번째 노드를 생성하기 위한 코드

```
node1 = Node()
node1.data = "다현"
print(node1.data, end = ' ')
```

실행 결과

```
다현
```

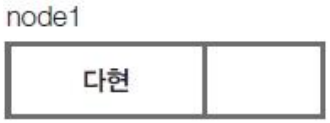


그림 4-9 첫 번째 노드 생성 결과

Section 02 단순 연결 리스트의 간단 구현

- 노드 연결
  - 두 번째 노드를 생성하고, 첫 번째 노드의 링크로 연결하는 코드

```
node2 = Node()
node2.data = "정연"
node1.link = node2    # 첫 번째 노드의 링크에 두 번째 노드를 넣어 연결
```

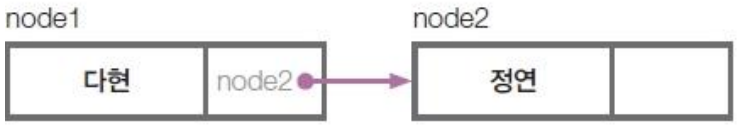


그림 4-10 두 번째 노드 생성 결과

Section 02 단순 연결 리스트의 간단 구현

- **데이터가 5개인 단순 연결 리스트 생성**

그림 4-11 생성할 단순 연결 리스트 예

Code04-01.py 데이터가 5개인 단순 연결 리스트 생성

```
1  class Node() :
2      def __init__ (self) :
3          self.data = None
4          self.link = None
5
6  node1 = Node()
7  node1.data = "다현"
8
9  node2 = Node()
10 node2.data = "정연"
11 node1.link = node2
12
13 node3 = Node()
14 node3.data = "쯔위"
15 node2.link = node3
```

Section 02 단순 연결 리스트의 간단 구현

```
16
17 node4 = Node()
18 node4.data = "사나"
19 node3.link = node4
20
21 node5 = Node()
22 node5.data = "지효"
23 node4.link = node5
24
25 print(node1.data, end = ' ')
26 print(node1.link.data, end = ' ')
27 print(node1.link.link.data, end = ' ')
28 print(node1.link.link.link.data, end = ' ')
29 print(node1.link.link.link.link.data, end = ' ')
```

실행 결과

다현 정연 쯔위 사나 지효

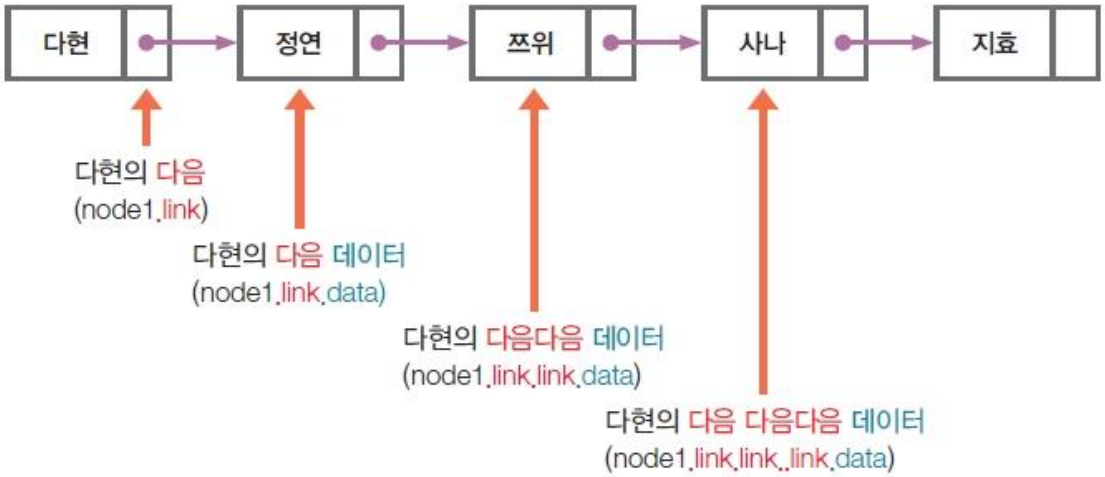
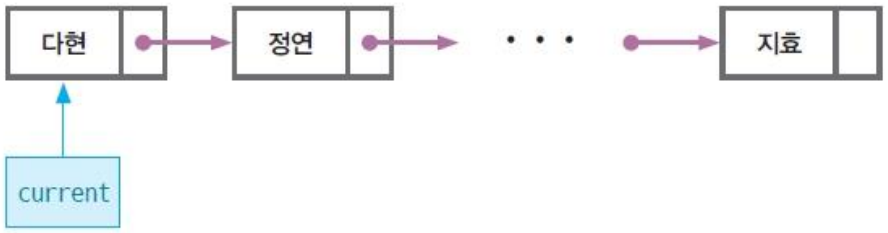


그림 4-12 첫 번째 노드부터 단순 연결 리스트의 노드 접근 방법

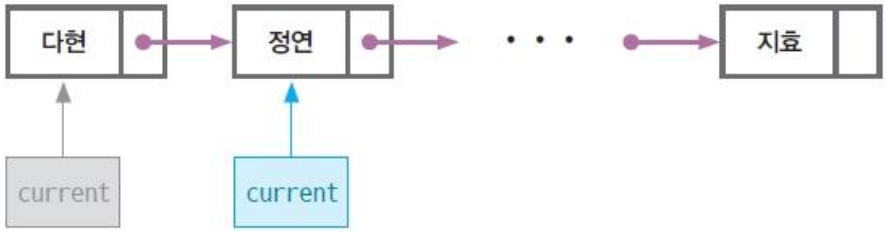
Section 02 단순 연결 리스트의 간단 구현

- 25~29행을 노드의 처음부터 끝까지 출력하는 함수로 작성하기 위한 동작 과정

❶ 첫 번째 노드를 현재(current) 노드로 지정하고, 현재 노드의 데이터인 다현을 출력한다.



❷ 현재 노드의 링크가 비어 있지 않다면 현재 노드를 현재 노드의 링크가 가리키는 노드로 변경한 후 현재 노드의 데이터인 정연을 출력한다.



❸ 앞의 ❷단계를 반복하다 현재 노드의 링크가 비어 있으면 종료한다.

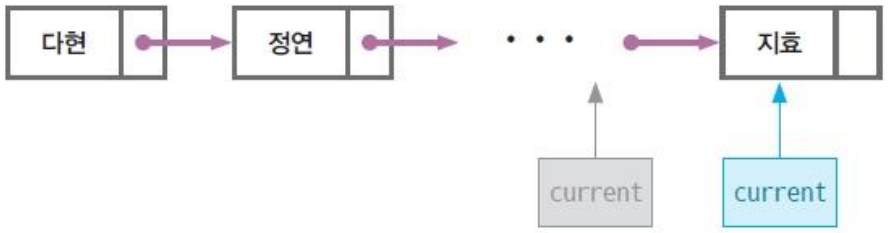


그림 4-13 단순 연결 리스트의 모든 노드 출력

Section 02 단순 연결 리스트의 간단 구현

- Code04-01.py의 25~29행을 Code04-02.py처럼 변경하여 단순 연결 리스트를 모두 출력

Code04-02.py 데이터가 5개인 단순 연결 리스트 생성(개선 버전)

```
… # 생략(Code04-01.py의 1~23행과 동일)
24
25 current = node1
26 print(current.data, end = ' ')
27 while current.link != None :
28     current = current.link
29     print(current.data, end = ' ')
```

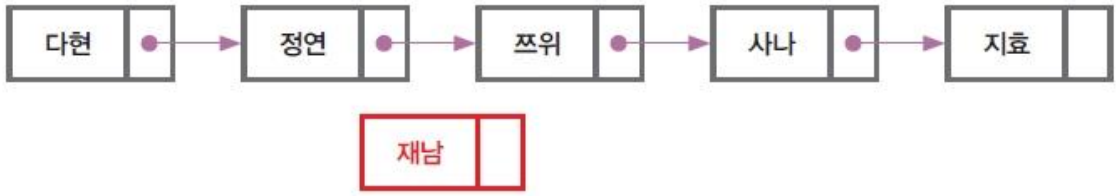
❶ ❷~❸

실행 결과

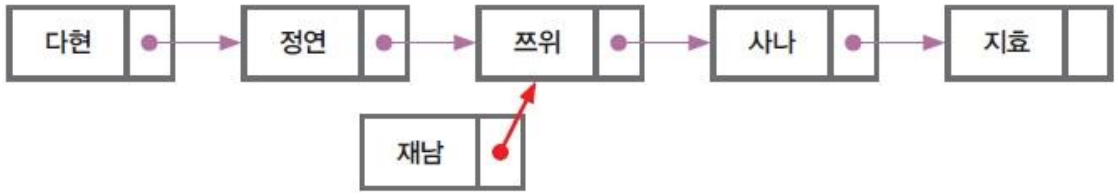
```
다현 정연 쯔위 사나 지효
```

- **노드 삽입 : 중간에 데이터 삽입**

❶ 새 노드를 생성하고 데이터에 재남을 입력한다.



❷ 새 노드(재남 노드)의 링크에 정연 노드의 링크를 복사한다. 그러면 정연 노드와 재남 노드 모두 쯔위 노드를 가리킨다.



❸ 정연 노드의 링크에 재남 노드를 지정한다.

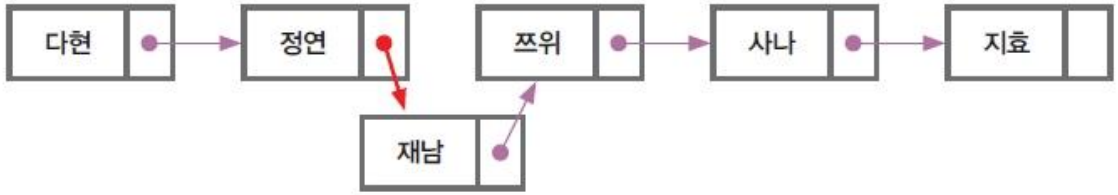


그림 4-14 단순 연결 리스트의 중간 노드 삽입

Section 02 단순 연결 리스트의 간단 구현

Code04-03.py 데이터가 5개인 단순 연결 리스트의 노드 삽입

```
... # 생략(Code04-01.py의 1~23행과 동일)
24
25 newNode = Node()
26 newNode.data = "재남"                    ❶
27 newNode.link = node2.link ❷ # 정연의 링크
28 node2.link = newNode ❸
29
... # 생략(Code04-02.py의 25~29행과 동일)
```

실행 결과

```
다현 정연 재남 쯔위 사나 지효
```

▪ 노드 삭제 : 중간 데이터 삭제

❶ 삭제할 쯔위 노드의 링크를 바로 앞 정연 노드의 링크로 복사한다. 그러면 정연 노드가 사나 노드를 가리킨다.

❷ 쯔위 노드를 삭제한다.

그림 4-15 단순 연결 리스트의 데이터 삭제

Code04-04.py 데이터가 5개인 단순 연결 리스트의 노드 삭제

```
… # 생략(Code04-01.py의 1~23행과 동일)
24
25 node2.link = node3.link ❶ # 쯔위의 링크를 정연의 링크로 복사
26 del(node3) ❷              # 쯔위 삭제
27
… # 생략(Code04-02.py의 25~29행과 동일)
```

실행 결과

```
다현 정연 사나 지효
```

단순 연결 리스트의 일반 형태

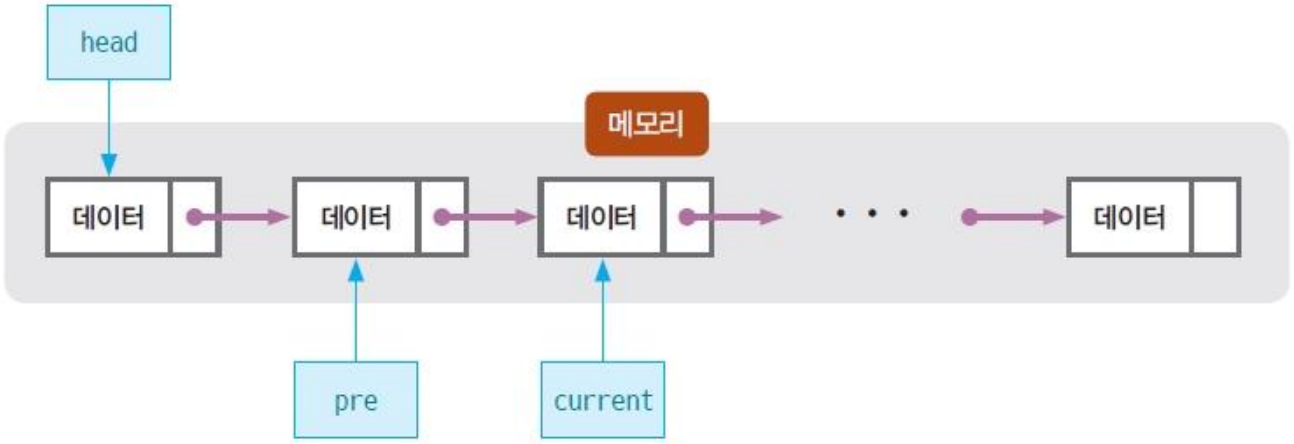


그림 4-16 단순 연결 리스트의 전체 구성 환경

- 헤드(head)는 첫 번째 노드를 가리키고, 현재(current)는 지금 처리 중인 노드를 가리키며, 이전(pre)은 현재 처리 중인 노드의 바로 앞 노드를 가리킴

- 처음에는 모두 비어 있으면 되므로 다음과 같이 초기화함

```
memory = []
head, current, pre = None, None, None
```

배열에 저장된 데이터 입력 과정

데이터 입력 과정

첫 번째 데이터 입력

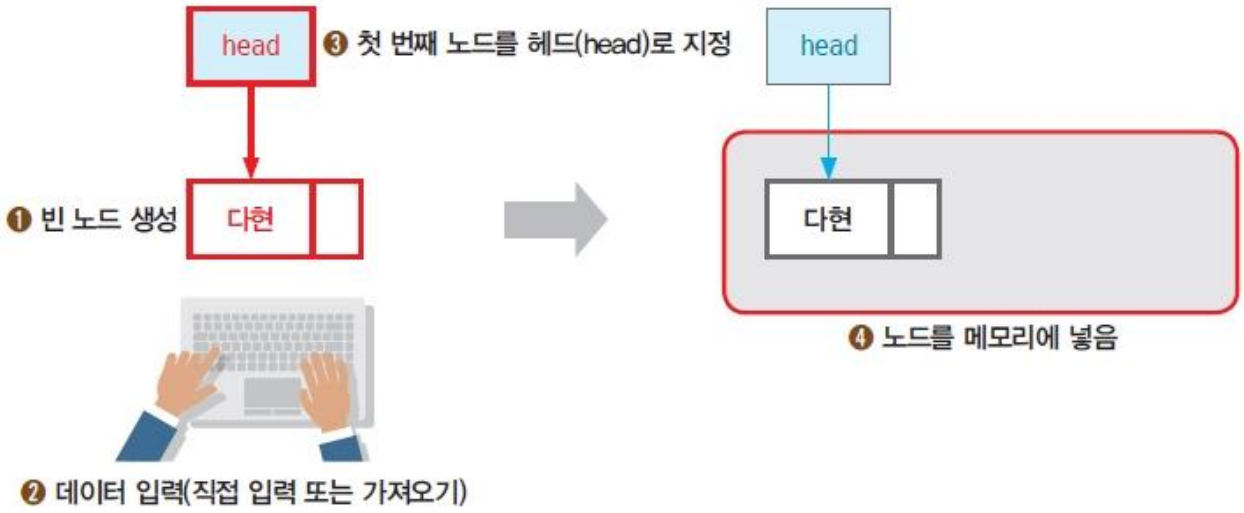


그림 4-17 단순 연결 리스트의 첫 번째 데이터 입력

```
❶ node = Node()
❷ node.data = dataArray[0]   # 첫 번째 노드
❸ head = node
❹ memory.append(node)
```

- 두 번째 이후 데이터 입력

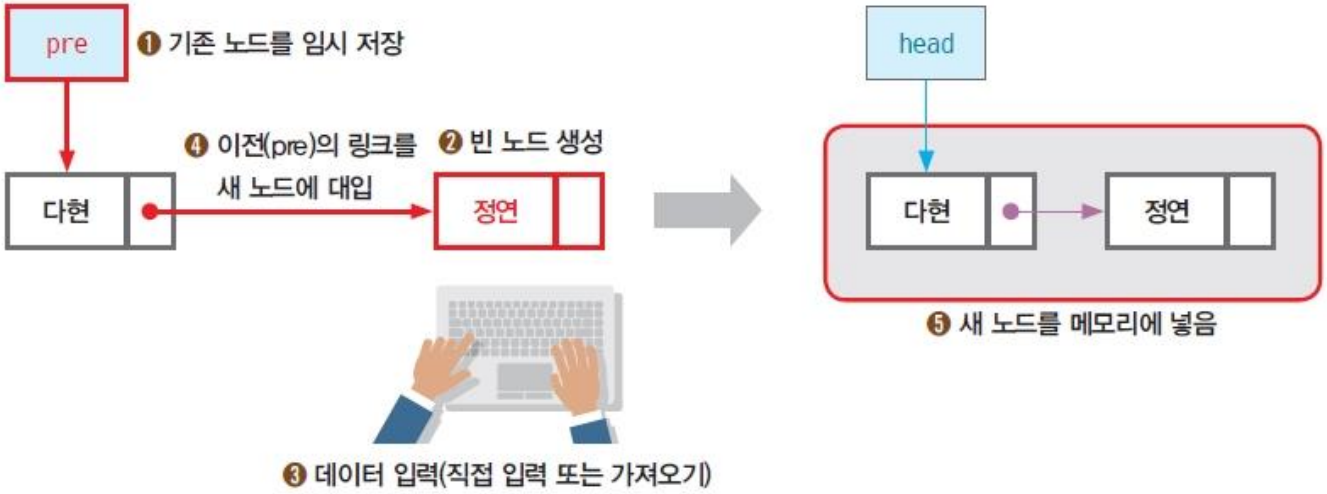


그림 4-18 단순 연결 리스트의 두 번째 이후 데이터 입력

```
❶ pre = node
❷ node = Node()
❸ node.data = data    # 두 번째 이후 노드
❹ preNode.link = node
❺ memory.append(node)
```

Section 03 단순 연결 리스트의 일반 구현

- 일반 단순 연결 리스트의 생성 함수 완성
  - 배열에 저장된 데이터를 모두 꺼내 단순 연결 리스트를 생성하는 예

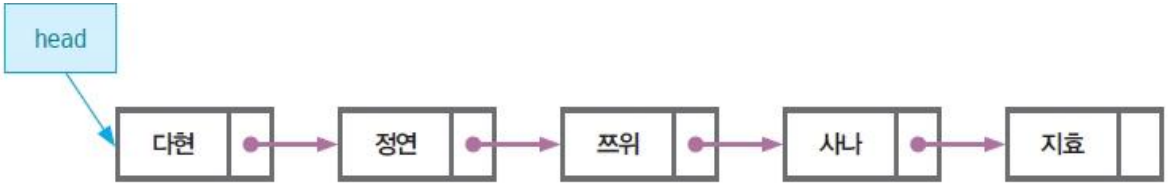


그림 4-19 배열을 이용하여 생성할 단순 연결 리스트 예

Code04-05.py 단순 연결 리스트 생성

```
## 클래스와 함수 선언 부분 ##
class Node() :
    def __init__ (self) :
        self.data = None
        self.link = None

def printNodes(start) :
    current = start
    if current == None :
        return
    print(current.data, end = ' ')
    while current.link != None :
        current = current.link
        print(current.data, end = ' ')
    print()

```

Section 03 단순 연결 리스트의 일반 구현

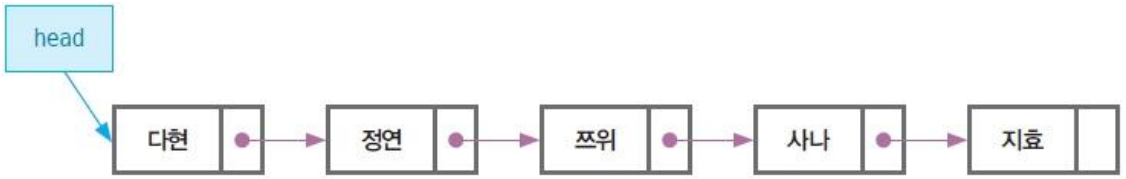
```
17 ## 전역 변수 선언 부분 ##
18 memory = []
19 head, current, pre = None, None, None
20 dataArray = ["다현", "정연", "쯔위", "사나", "지효"]
21
22 ## 메인 코드 부분 ##
23 if __name__ == "__main__" :
24
25     node = Node()                    # 첫 번째 노드
26     node.data = dataArray[0]
27     head = node
28     memory.append(node)
29
30     for data in dataArray[1:] :      # 두 번째 이후 노드
31         pre = node
32         node = Node()
33         node.data = data
34         pre.link = node
35         memory.append(node)
36
37     printNodes(head)
```

실행 결과

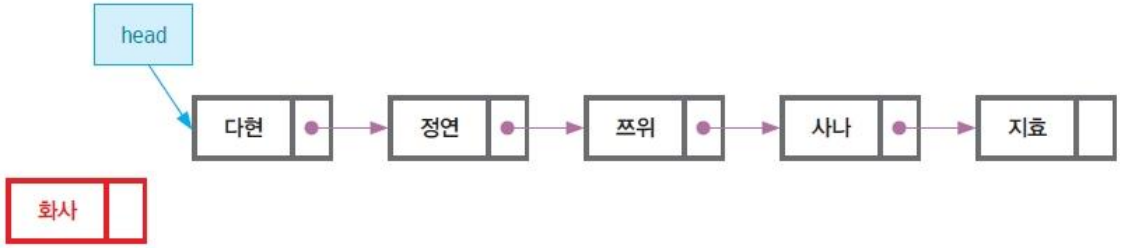
다현 정연 쯔위 사나 지효

▪ 노드 삽입 : 첫 번째 노드 삽입

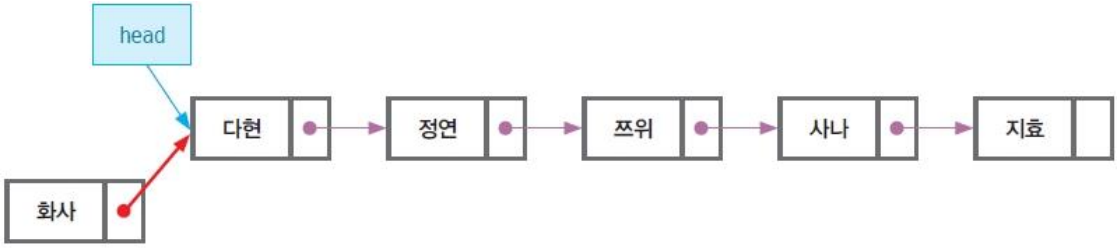
0 맨 앞에 노드를 삽입하기 전 초기 상태



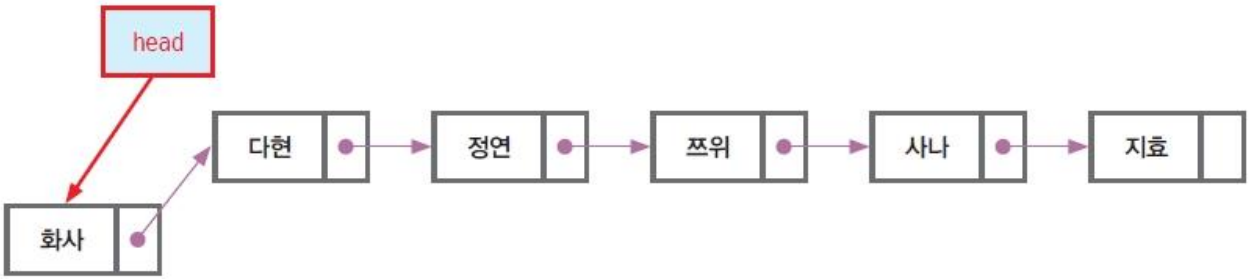
1 새 노드(화사 노드)를 생성한다.



2 새 노드의 링크로 헤드(head) 노드가 가리키는 노드를 지정한다.



3 헤드 노드를 새 노드로 지정한다.

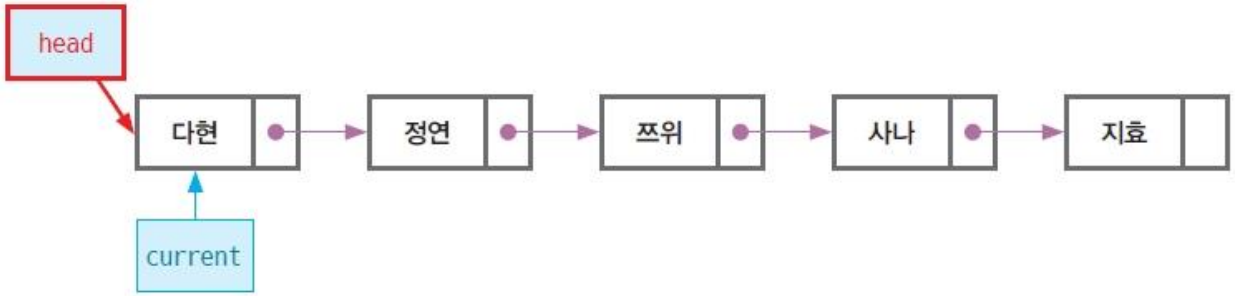



```
1 node = Node()
  node.data = "화사"
2 node.link = head
3 head = node
```

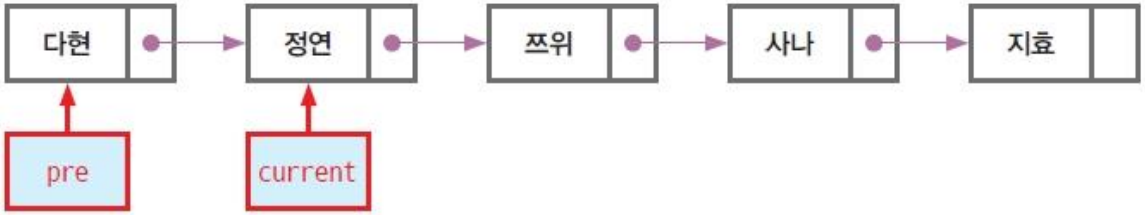
▪ 노드 삽입 : 중간 노드 삽입

0 중간 노드 삽입 전 초기 상태

1 헤드(head)에서 시작해서 현재(current) 노드가 사나인지 확인한다.

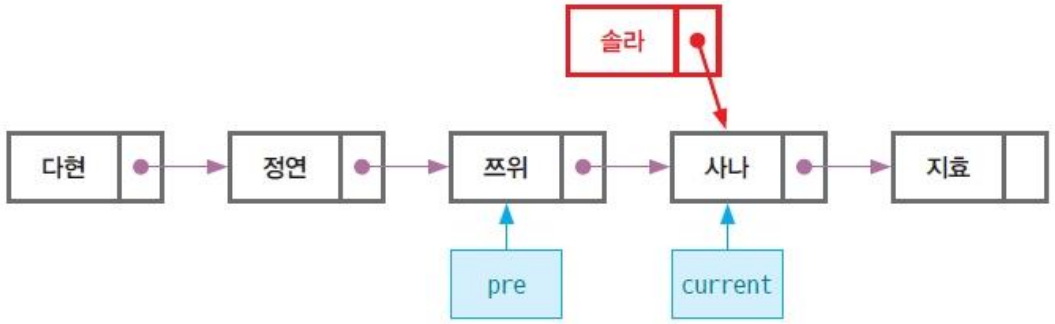


2 현재 노드를 이전(pre) 노드로 지정하고, 현재 노드를 다음 노드로 이동한다. 그리고 현재 노드가 사나인지 확인한다.

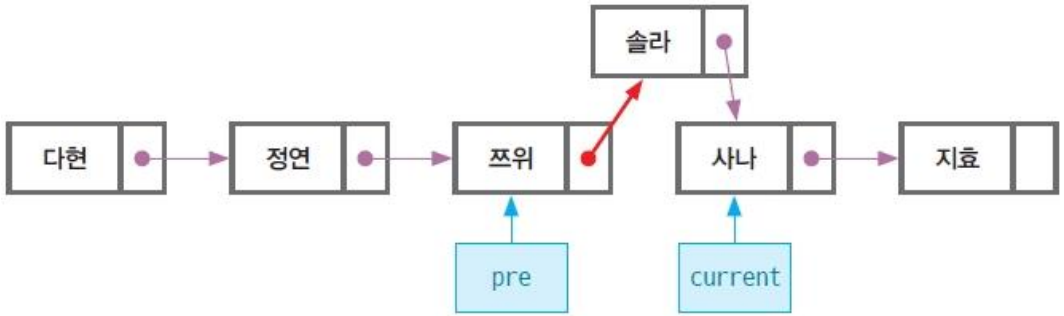


3 현재 노드가 사나일 때까지 2 단계를 반복한다.

4 현재 노드가 사나라면 우선 새 노드(솔라 노드)를 생성한 후 이전 노드의 링크를 새 노드의 링크로 지정한다.



5 이전 노드의 링크를 새 노드로 지정한다.

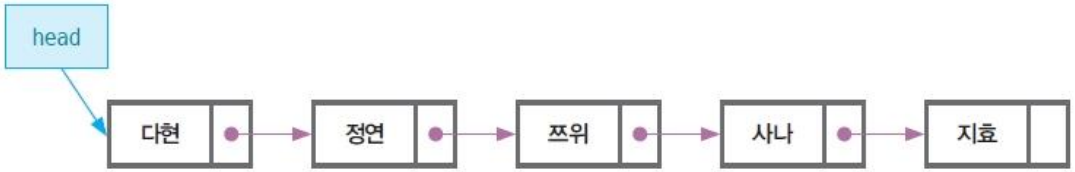


Section 03 단순 연결 리스트의 일반 구현

```
1 current = head
  while 마지막 노드까지 :
2     pre = current
      current = current.link
3     if current.data == "사나" :
4         node = Node()
          node.data = "솔라"
          node.link = current
5         pre.link = node
```

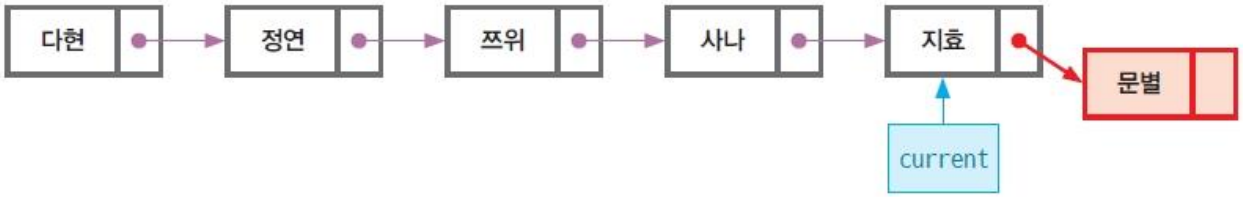
▪ 노드 삽입 : 마지막 노드 삽입

0 마지막 노드 삽입 전 초기 상태



1~3 중간 노드 삽입 과정과 동일하므로 없는 데이터인 재남을 찾는다.

4 마지막 노드까지 재남을 찾지 못했다면 우선 새 노드(문별 노드)를 생성한 후 현재(current) 노드의 링크를 새 노드로 지정한다.




```
1~3 # 마지막 노드까지 "재남"을 찾지 못한 후
    node = Node()
4   node.data = "문별"
    current.link = node
```

Section 03 단순 연결 리스트의 일반 구현

- 노드 삽입 함수의 완성

Code04-06.py 단순 연결 리스트의 노드 삽입 함수

```
1  ## 클래스와 함수 선언 부분 ##
2  class Node() :
…      # 생략(Code04-05.py의 3~5행과 동일)
6
7  def printNodes(start) :
…      # 생략(Code04-05.py의 8~15행과 동일)
16
17 def insertNode(findData, insertData) :
18     global memory, head, current, pre
19
20     if head.data == findData :      # 첫 번째 노드 삽입
21         node = Node()
22         node.data = insertData
23         node.link = head
24         head = node
25         return
26
27     current = head
28     while current.link != None :    # 중간 노드 삽입
29         pre = current
30         current = current.link
```

Section 03 단순 연결 리스트의 일반 구현

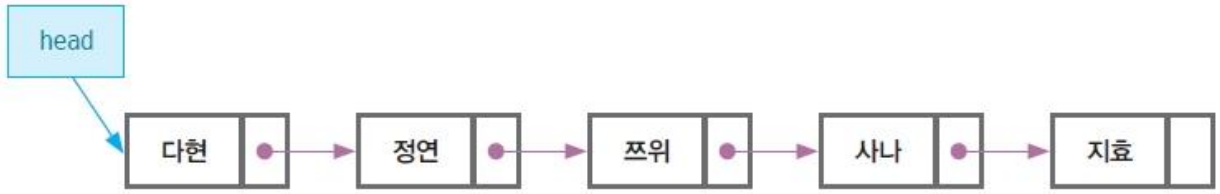
```
31          if current.data == findData :
32              node = Node()
33              node.data = insertData
34              node.link = current
35              pre.link = node
36              return
37
38      node = Node()          # 마지막 노드 삽입
39      node.data = insertData
40      current.link = node
41
42  ## 전역 변수 선언 부분 ##
…   # 생략(Code04-05.py의 18~20행과 동일)
46
47  ## 메인 코드 부분 ##
48  if __name__ == "__main__" :
49
…       # 생략(Code04-05.py의 25~37행과 동일)
63
64      insertNode("다현", "화사")
65      printNodes(head)
66
67      insertNode("사나", "솔라")
68      printNodes(head)
69
70      insertNode("재남", "문별")
71      printNodes(head)
```

실행 결과

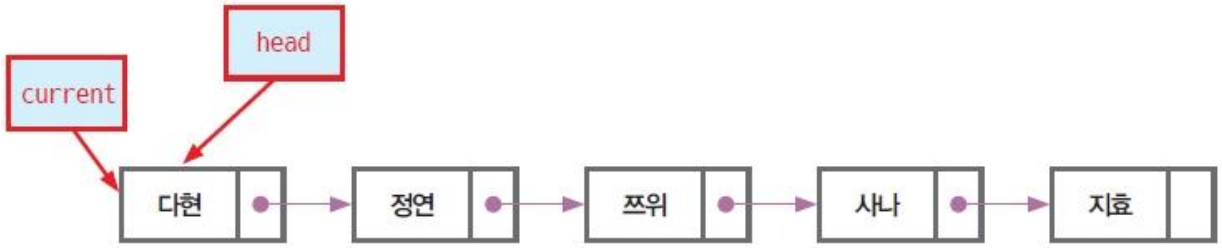
```
다현 정연 쯔위 사나 지효 → 초기상태
화사 다현 정연 쯔위 사나 지효
화사 다현 정연 쯔위 솔라 사나 지효
화사 다현 정연 쯔위 솔라 사나 지효 문별
```

▪ 노드 삭제 : 첫 번째 노드 삭제

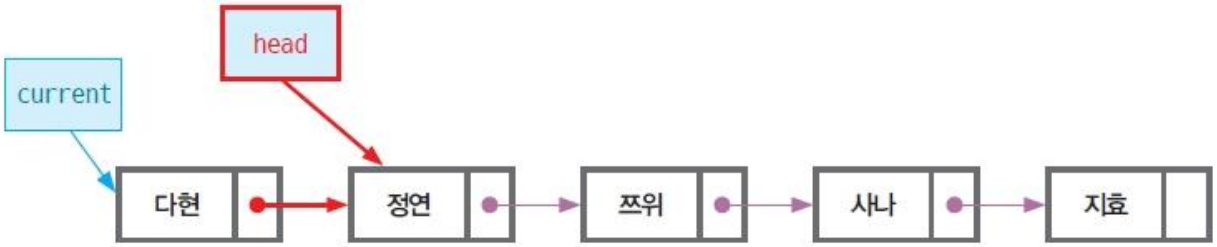
0 노드 삭제 전 초기 상태



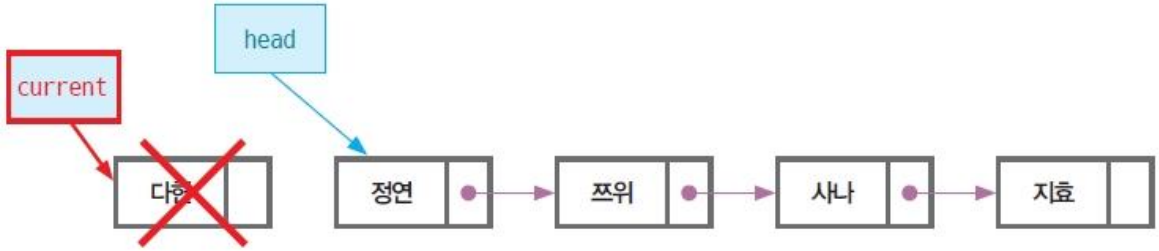
1 현재 노드(current)를 삭제할 노드인 헤드(head)와 동일하게 만든다.



2 헤드를 삭제할 노드(다현 노드)의 링크가 가리키던 정연 노드로 변경된다.



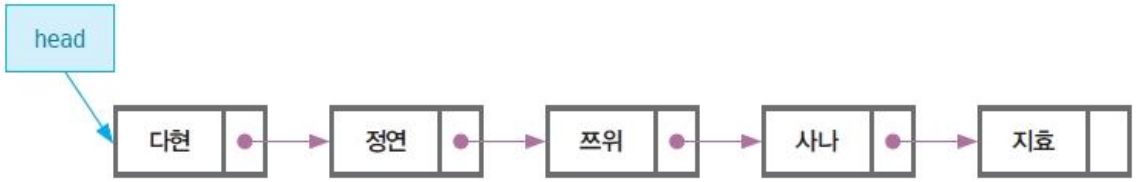
3 현재 노드를 메모리에서 제거한다.



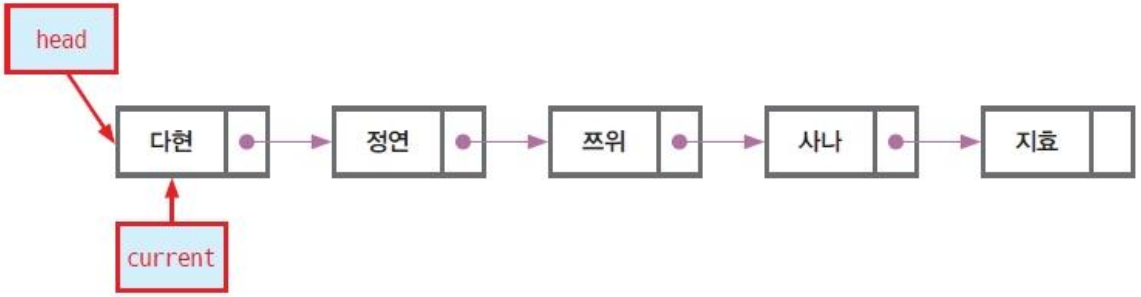

```
1 current = head
2 head = head.link
3 del(current)
```

▪ 노드 삭제 : 첫 번째 외 노드 삭제

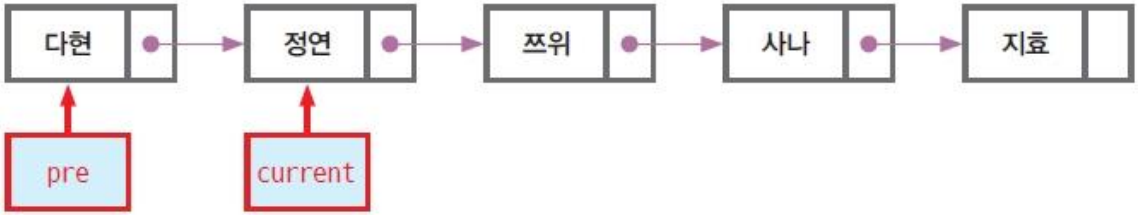
0 노드 삭제 전 초기 상태



1 헤드(head)에서 시작해서 현재 노드(current)가 쯔위인지 확인한다.



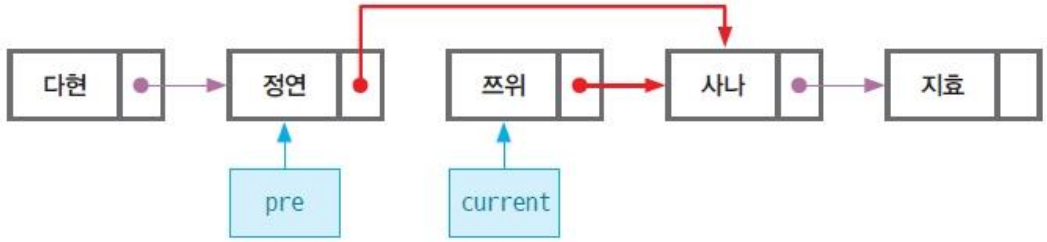
2 현재 노드를 이전 노드(pre)로 저장하고, 현재 노드를 다음 노드로 이동한다.
그리고 현재 노드가 쯔위인지 확인한다.



Section 03 단순 연결 리스트의 일반 구현

3 현재 노드가 쯔위일 때까지 2단계를 반복한다.

4 현재 노드가 쯔위라면, 이전 노드의 링크를 현재 노드의 링크로 지정한다.



5 현재 노드를 메모리에서 삭제한다.


```
1 current = head
3 while 마지막 노드까지 :
    2 pre = current
      current = current.link
    4 if current.data == "쯔위" :
        pre.link = current.link
    5 del(current)
```

- 노드 삭제 함수의 완성

Code04-07.py 단순 연결 리스트의 노드 삭제 함수

```
1  ## 클래스와 함수 선언 부분 ##
2  class Node() :
…      # 생략(Code04-05.py의 3~5행과 동일)
6
7  def printNodes(start) :
…      # 생략(Code04-05.py의 8~15행과 동일)
16
17 def deleteNode(deleteData) :
18     global memory, head, current, pre
19
20     if head.data == deleteData :      # 첫 번째 노드 삭제
21         current = head
22         head = head.link
23         del(current)
24         return
25
26     current = head                    # 첫 번째 외 노드 삭제
27     while current.link != None :
28         pre = current
```

```
29          current = current.link
30          if current.data == deleteData :
31              pre.link = current.link
32              del(current)
33              return
34
35  ## 전역 변수 선언 부분 ##
…   # 생략(Code04-05.py의 18~20행과 동일)
39
40  ## 메인 코드 부분 ##
41  if __name__ == "__main__" :
42
…       # 생략(Code04-05.py의 25~37행과 동일)
56
57      deleteNode("다현")
58      printNodes(head)
59
60      deleteNode("쯔위")
61      printNodes(head)
62
63      deleteNode("지효")
64      printNodes(head)
65
66      deleteNode("재남")
67      printNodes(head)
```

실행 결과

```
다현 정연 쯔위 사나 지효 → 초기상태
정연 쯔위 사나 지효
정연 사나 지효
정연 사나
정연 사나
```

SELF STUDY 4-1

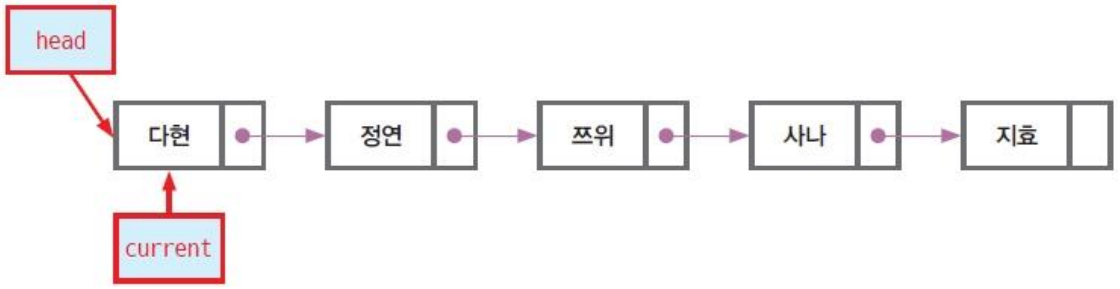
Code04-07.py를 수정해서 데이터가 삭제되면서 삭제되는 위치가 출력되도록 하자. 예를 들어 첫 번째 노드가 삭제되면 '첫 노드가 삭제됨'이, 중간 노드가 삭제되면 '중간 노드가 삭제됨'이, 삭제된 노드가 없으면 '삭제된 노드가 없음'이 출력되도록 한다.

실행 결과

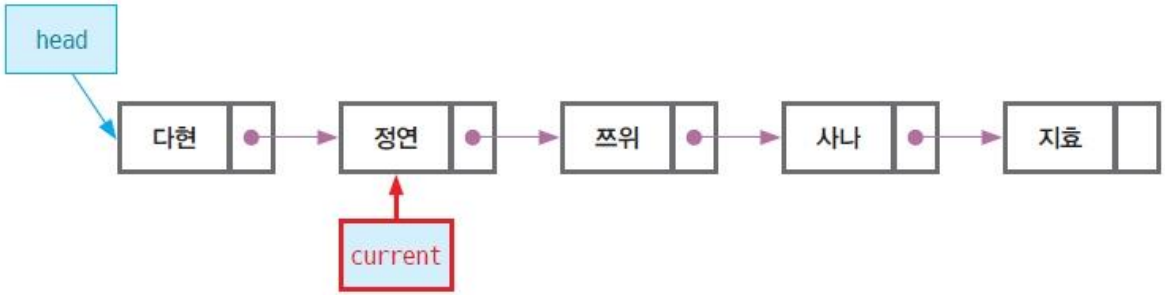
```
다현 정연 쯔위 사나 지효
# 첫 노드가 삭제됨 #
정연 쯔위 사나 지효
# 중간 노드가 삭제됨 #
정연 사나 지효
# 중간 노드가 삭제됨 #
정연 지효
# 삭제된 노드가 없음 #
정연 지효
```

노드 검색

1 현재 노드(current)를 첫 번째 노드인 헤드(head)와 동일하게 만들고, 현재 노드가 검색할 데이터인지 비교한다. 검색할 데이터와 동일하다면 현재 노드를 반환한다.



2 현재 노드를 다음 노드로 이동하고, 검색할 데이터와 동일하다면 현재 노드를 반환한다.



3 앞의 2단계를 끝까지 진행하고, 검색할 데이터를 찾지 못했다면 None을 반환한다.

Code04-08.py 단순 연결 리스트의 노드 검색 함수

```
## 클래스와 함수 선언 부분 ##
class Node() :
    def __init__ (self) :
        self.data = None
        self.link = None

def printNodes(start) :
    current = start
    if current == None :
        return
    print(current.data, end = ' ')
    while current.link != None :
        current = current.link
        print(current.data, end = ' ')
    print()

def findNode(findData) :
    global memory, head, current, pre

```

```
20      current = head
21      if current.data == findData :
22          return current
23      while current.link != None :
24          current = current.link
25          if current.data == findData :
26              return current
27      return Node()    # 빈 노드 반환
28
29  ## 전역 변수 선언 부분 ##
30  memory = []
31  head, current, pre = None, None, None
32  dataArray = ["다현", "정연", "쯔위", "사나", "지효"]
33
34  ## 메인 코드 부분 ##
35  if __name__ == "__main__" :
36
37      node = Node()          # 첫 번째 노드
38      node.data = dataArray[0]
39      head = node
40      memory.append(node)
```

```
41
42      for data in dataArray[1:] :   # 두 번째 이후 노드
43          pre = node
44          node = Node()
45          node.data = data
46          pre.link = node
47          memory.append(node)
48
49      printNodes(head)
50
51      fNode = findNode("다현")
52      print(fNode.data)
53
54      fNode = findNode("쯔위")
55      print(fNode.data)
56
57      fNode = findNode("재남")
58      print(fNode.data)
```

실행 결과

```
다현 정연 쯔위 사나 지효 → 초기상태
다현
쯔위
None
```

단순 연결 리스트를 응용하는 프로그램 예

- 이름과 연락처를 무작위로 입력하면 단순 연결 리스트에 이름 순서대로 저장됨

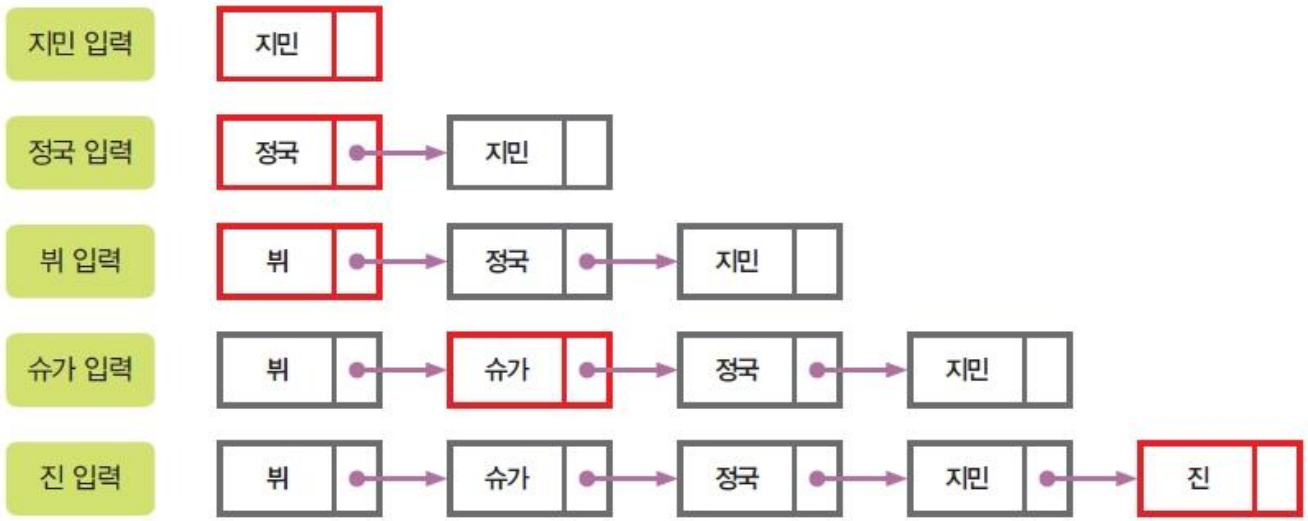
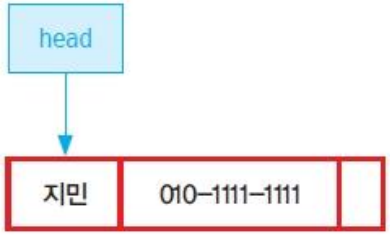


그림 4-20 이름 순서로 정렬된 단순 연결 리스트

1 입력할 데이터를 2차원 배열에 저장한다.

```
dataArray = [["지민", "010-1111-1111"], ["정국", "010-2222-2222"], ["뷔", "010-3333-3333"],
["슈가", "010-4444-4444"], ["진", "010-5555-5555"]]
```

2 데이터를 차례대로 가져온다. 먼저 지민을 단순 연결 리스트로 생성하자. 헤드(head)가 비어 있을 때는 헤드에 첫 노드를 지정한다.



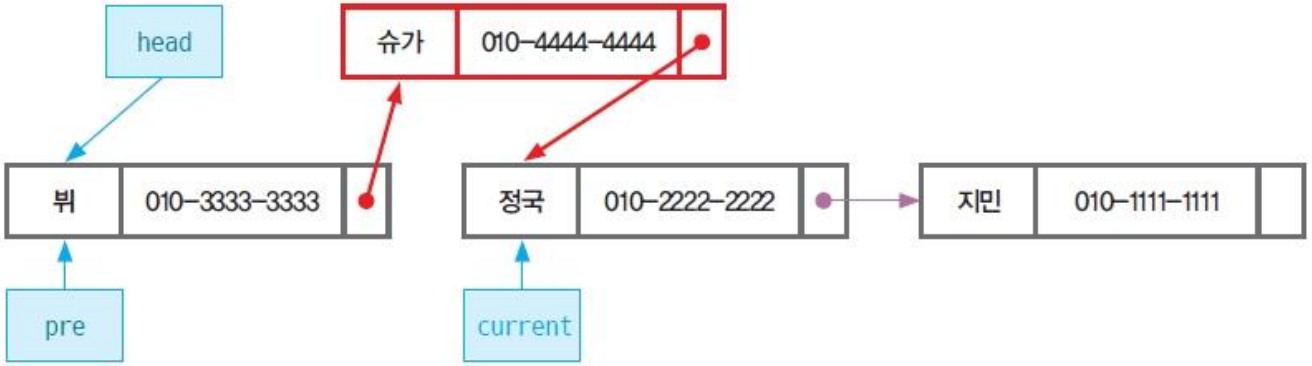

```
head = node
```

3 두 번째 정국을 대입한다. 새로운 데이터가 첫 노드보다 작다면 새 노드의 링크에 첫 노드를 입력하고, 헤드에는 새 노드를 지정한다.


```
node.link = head
head = node
```

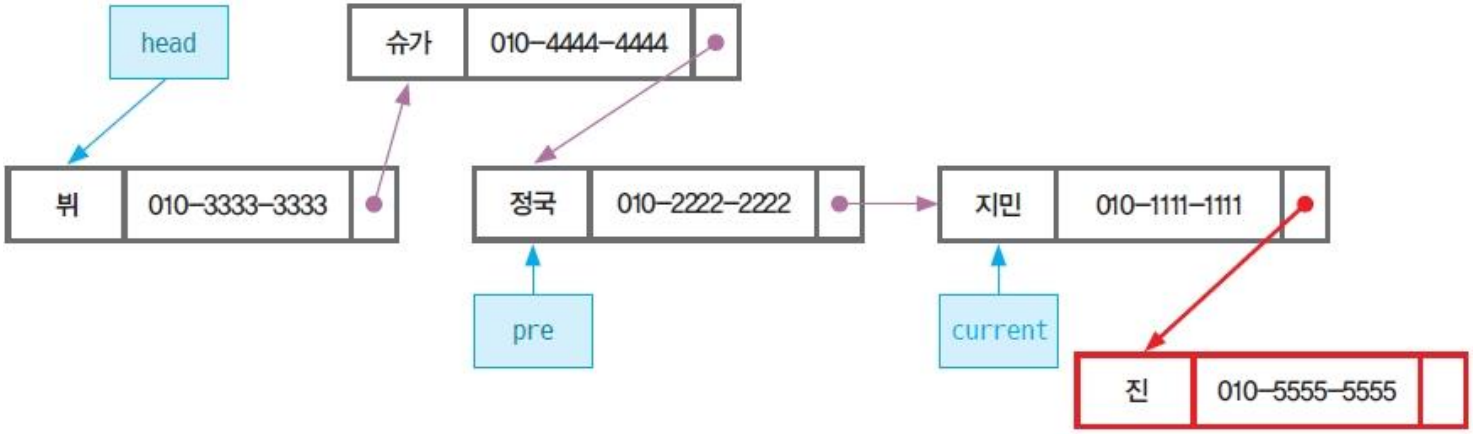
4 다음으로 뷔를 대입한다. 역시 새로운 데이터가 첫 노드보다 작다면 새 노드의 링크에 첫 노드를 입력하고, 헤드에는 새 노드를 지정한다.

5 다음으로 슈가를 입력한다. 슈가는 첫 노드보다 크기 때문에 중간에 삽입된다. 현재 노드(current)와 이전 노드(pre)를 이용해서 현재 노드가 슈가보다 크다면 이전 노드 링크를 슈가로 지정하고, 슈가 노드의 링크를 현재 노드로 지정한다.



```
current = head
while 노드의 끝까지 :
    pre = current
    current = current.link
    if 현재노드 > 입력할노드 :
        pre.link = node
        node.link = current
```

6 마지막으로 진을 입력한다. 진은 마지막 노드보다 크므로 마지막에 추가한다.




```
current.link = node
```

Section 04 단순 연결 리스트의 응용

- 앞의 과정을 코드로 구현

Code04-09.py 단순 연결 리스트를 활용한 명함 관리 프로그램

```
1  ## 클래스와 함수 선언 부분 ##
2  class Node() :
3      def __init__ (self) :
4          self.data = None
5          self.link = None
6
7  def printNodes(start) :
8      current = start
9      if current == None :
10         return
11     print(current.data, end = ' ')
12     while current.link != None :
13         current = current.link
14         print(current.data, end = ' ')
15     print()
16
17 def makeSimpleLinkedList(namePhone) :
18     global memory, head, current, pre
19     printNodes(head)
20
```

```
21      node = Node()
22      node.data = namePhone
23      memory.append(node)
24      if head == None :               # 첫 번째 노드일 때
25          head = node
26          return
27
28      if head.data[0] > namePhone[0] :  # 첫 번째 노드보다 작을 때
29          node.link = head
30          head = node
31          return
32
33      # 중간 노드로 삽입하는 경우
34      current = head
35      while current.link != None :
36          pre = current
37          current = current.link
38          if current.data[0] > namePhone[0] :
39              pre.link = node
40              node.link = current
41              return
42
43      # 삽입하는 노드가 가장 큰 경우
44      current.link = node
45
```

```
46 ## 전역 변수 선언 부분 ##
47 memory = []
48 head, current, pre = None, None, None
49 dataArray = [["지민", "010-1111-1111"], ["정국", "010-2222-2222"], ["뷔", "010-3333-3333"],
   ["슈가", "010-4444-4444"], ["진", "010-5555-5555"]]
50
51 ## 메인 코드 부분 ##
52 if __name__ == "__main__" :
53
54     for data in dataArray :
55         makeSimpleLinkedList(data)
56
57     printNodes(head)
```

실행 결과

```
['지민', '010-1111-1111']
['정국', '010-2222-2222']['지민', '010-1111-1111']
['뷔', '010-3333-3333']['정국', '010-2222-2222']['지민', '010-1111-1111']
['뷔', '010-3333-3333']['슈가', '010-4444-4444']['정국', '010-2222-2222']['지민', '010-1111-1111']
['뷔', '010-3333-3333']['슈가', '010-4444-4444']['정국', '010-2222-2222']['지민', '010-1111-
1111']['진', '010-5555-5555']
```

Section 04 단순 연결 리스트의 응용

SELF STUDY 4-2

Code04-09.py를 수정해서 전화번호 대신에 dataArray에 키를 사용하자. 그리고 키 순서대로 단순 연결 리스트를 생성하자.

dataArray = [["지민", 180], ["정국", 177], ["뷔", 183], ["슈가", 175], ["진", 179]]

실행 결과

```
['지민', 180]
['정국', 177] ['지민', 180]
['정국', 177] ['지민', 180] ['뷔', 183]
['슈가', 175] ['정국', 177] ['지민', 180] ['뷔', 183]
['슈가', 175] ['정국', 177] ['진', 179] ['지민', 180] ['뷔', 183]
```

응용예제 01 사용자가 입력한 정보 관리하기

난이도★★☆☆☆

예제 설명 사용자가 이름과 이메일을 입력하면 이메일 순서대로 단순 연결 리스트를 생성하는 프로그램을 작성한다. 이름에서 그냥 Enter를 누르면 입력을 종료한다.

실행 결과

응용예제 02 로또 추첨하기

난이도★★☆☆☆

예제 설명 1~45 숫자 6개를 뽑는 로또 추첨 프로그램을 작성한다. 뽑은 숫자는 순서대로 단순 연결 리스트로 저장한다.

실행 결과