

Data Structure

# 자료구조



## 연결된 구조



한국기술교육대학교  
온라인평생교육원

## 학습내용

- 연결된 구조란?
- 단순 연결 구조 응용: 연결된 스택
- 단순 연결 구조 응용: 연결 리스트

## 학습목표

- 배열 구조와 연결된 구조의 특징을 설명할 수 있다.
- 노드의 개념과 단순 연결 리스트의 구조를 설명할 수 있다.
- 단순 연결 리스트로 스택과 리스트를 구현할 수 있다.

# 연결된 구조란?

연결된 구조

연결된 구조란?

## 1 배열 구조와 연결된 구조

### 배열 구조

- 항목들이 메모리에 연속적으로 저장되어 있음
- 시작 항목의 주소를 알면  $i$ 번째 항목의 위치를 바로 계산
- 항목 접근의 시간 복잡도:  $O(1)$
- 인덱스 연산이 가능  
예 A[3]
- 삽입/삭제 연산에서 많은 항목의 이동이 발생



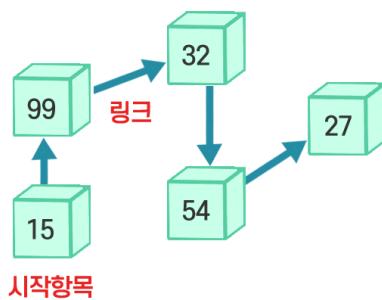
연결된 구조

연결된 구조란?

## 1 배열 구조와 연결된 구조

### 연결된 구조

- 배열과 달리 항목들이 메모리에 연속적으로 저장되어 있지 않음
- 다음 자료를 찾아가려면? “링크”가 필요함
- 데이터와 링크로 이루어진 “노드”가 연결된 구조를 만듦
- 시작 항목을 알더라도  $i$ 번째 항목의 위치를 바로 계산할 수 없음
  - 링크를 따라 하나씩 따라 가야 함



# 연결된 구조란?

연결된 구조

연결된 구조란?

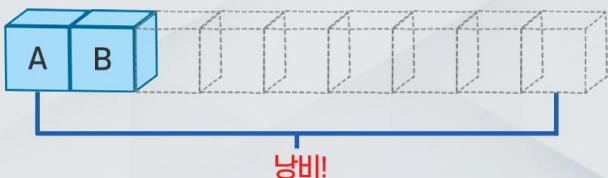
## 1 배열 구조와 연결된 구조

### ● 연결된 구조의 특징

용량이 고정되지 않음

- 메모리의 낭비가 없음

배열 구조



연결된 구조



연결된 구조

연결된 구조란?

## 1 배열 구조와 연결된 구조

### ● 연결된 구조의 특징

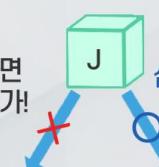
용량이 고정되지 않음

- 크기의 제한도 없음

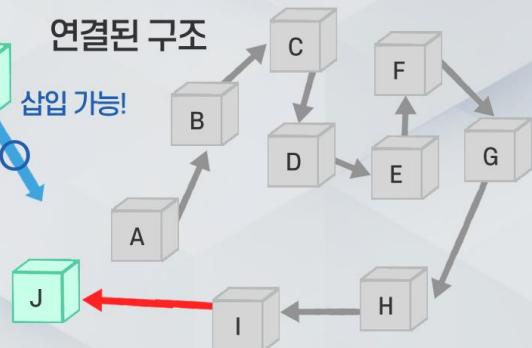
배열 구조



가득 차면  
삽입 불가!



연결된 구조





# 연결된 구조란?

연결된 구조

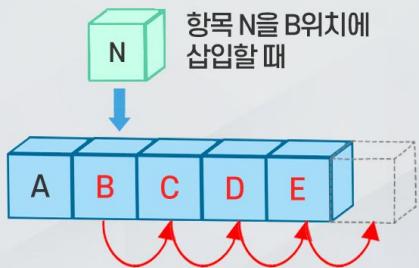
연결된 구조란?

## 1 배열 구조와 연결된 구조

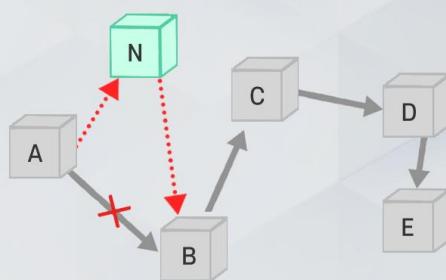
### ● 연결된 구조의 특징

중간에 자료를 삽입하는 과정이 효율적

배열 구조



연결된 구조



연결된 구조

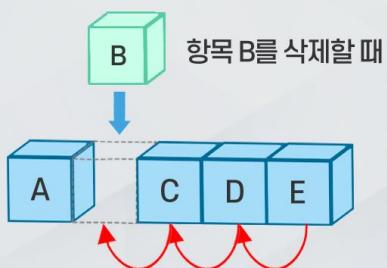
연결된 구조란?

## 1 배열 구조와 연결된 구조

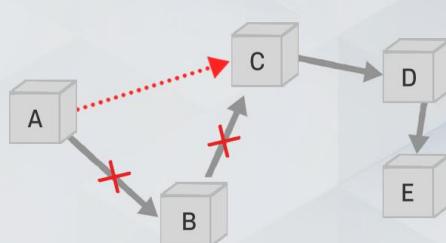
### ● 연결된 구조의 특징

중간에 자료를 삭제하는 과정이 효율적

배열 구조



연결된 구조





# 연결된 구조란?

연결된 구조

연결된 구조란?

## 1 배열 구조와 연결된 구조

### ● 연결된 구조의 특징

#### 연결된 구조의 단점

- 구현이 복잡
- N번째 항목에 접근하는데  $O(n)$ 의 시간이 걸림

연결된 구조

연결된 구조란?

## 2 노드(Node)

#### 노드(Node)

- 연결된 구조에서 하나의 상자
- 데이터 필드(data field)
- 하나 이상의 링크 필드(link field)
  - 포인터(pointer), 또는 참조자(reference)의 개념



# 연결된 구조란?

연결된 구조

연결된 구조란?

## 2 노드(Node)

### ◉ 링크의 개념

#### 링크(link)

- 일종의 포인터(pointer)
- 포인터: 다른 객체를 가리키고(pointing) 있는 변수
- 다른 노드를 가리킴
  - 예 리스트에서 다음 노드, 이전 노드 등
  - 예 이진 트리에서 왼쪽 자식 노드, 오른쪽 자식 노드 등

- 파이썬에서는 모든 변수가 포인터
- 파이썬의 변수는 값을 가지지 않고, 가리키기만(참조) 함

연결된 구조

연결된 구조란?

## 2 노드(Node)

### ◉ 헤드 포인터

#### 헤드 포인터(head pointer)

- 연결된 구조에서 자료구조 객체의 유일한 정보
- 시작 노드를 가리키는 포인터
- 마지막 노드: 다음 노드가 없으므로 링크가 None
  - 예 연결된 구조의 리스트(연결 리스트)



헤드 포인터  
(head pointer)

27

35

99

15

None

# 연결된 구조란?

연결된 구조

연결된 구조란?

## 2 노드(Node)

### ● 노드의 종류

1

#### 단순 연결 노드(Node)

##### ▶ 하나의 링크 필드(link field)



```
# Singly Linked Node.
class Node:
    def __init__(self, elem, link):
        self.data = elem
        self.link = link
```

연결된 구조

연결된 구조란?

## 2 노드(Node)

### ● 노드의 종류

2

#### 이중 연결 노드(DNode)

##### ▶ 링크 필드가 두 개(previous, next)



```
# Doubly Linked Node.
class DNode:
    def __init__(self, elem, prev, next):
        self.data = elem
        self.prev = prev
        self.next = next
```



# 연결된 구조란?

연결된 구조

연결된 구조란?

## 3 노드와 연결된 리스트의 종류

- 단순 연결 노드(Node) 사용

단순 연결 리스트(singly linked list)



원형 연결 리스트(circular linked list)



연결된 구조

연결된 구조란?

## 3 노드와 연결된 리스트의 종류

- 단순 연결 노드(Node) 사용

이중 연결 리스트(doubly linked list)



# 연결된 구조란?

연결된 구조

연결된 구조란?

## 2 노드(Node)

### ● 노드의 종류

1

#### 단순 연결 노드(Node)

##### ▶ 하나의 링크 필드(link field)



```
# Singly Linked Node.
class Node:
    def __init__(self, elem, link):
        self.data = elem
        self.link = link
```

연결된 구조

연결된 구조란?

## 2 노드(Node)

### ● 노드의 종류

2

#### 이중 연결 노드(DNode)

##### ▶ 링크 필드가 두 개(previous, next)



```
# Doubly Linked Node.
class DNode:
    def __init__(self, elem, prev, next):
        self.data = elem
        self.prev = prev
        self.next = next
```



# 연결된 구조란?

연결된 구조

연결된 구조란?

## 3 노드와 연결된 리스트의 종류

- 단순 연결 노드(Node) 사용

단순 연결 리스트(singly linked list)



원형 연결 리스트(circular linked list)



연결된 구조

연결된 구조란?

## 3 노드와 연결된 리스트의 종류

- 단순 연결 노드(Node) 사용

이중 연결 리스트(doubly linked list)



# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결된 스택

#### 1 연결된 스택의 구조

##### ● 스택의 추상 자료형

데이터	연산
후입 선출(LIFO)의 접근 방법을 유지하는 항목들의 모음	<ul style="list-style-type: none"> <li>▪ <code>Stack()</code>: 비어 있는 새로운 스택을 만듦</li> <li>▪ <code>isEmpty()</code>: 스택이 공백상태면 True를 아니면 False를 반환</li> <li>▪ <code>isFull()</code>: 스택이 포화상태이면 True를 아니면 False를 반환</li> <li>▪ <code>push(e)</code>: 항목 e를 스택의 맨 위에 추가</li> <li>▪ <code>pop()</code>: 스택의 상단에 있는 항목을 꺼내 반환</li> <li>▪ <code>peek()</code>: 스택의 상단에 있는 항목을 삭제하지 않고 반환</li> <li>▪ <code>size()</code>: 모든 항목들의 개수를 반환</li> <li>▪ <code>clear()</code>: 스택을 공백상태로 만듦</li> <li>▪ <code>display()</code>: 스택을 화면에 보기 좋게 출력</li> </ul>

## 연결된 구조

### 단순 연결 구조 응용 : 연결된 스택

#### 1 연결된 스택의 구조

연결된 스택
<ul style="list-style-type: none"> <li>▪ 클래스 이름: <code>LinkedStack</code></li> <li>▪ 구조           <ul style="list-style-type: none"> <li>배열 구조의 스택</li> <li>연결된 구조의 스택</li> </ul> </li> <li>▪ 데이터 멤버           <ul style="list-style-type: none"> <li>• <code>top</code>: 시작 노드(최근 삽입된 노드)를 가리키는 포인터</li> </ul> </li> </ul>



# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 2 연결된 스택의 구현

#### ● 클래스 정의와 멤버 변수

- ▶ 클래스 이름: LinkedStack
- ▶ top: 시작 노드(최근에 삽입된 노드)를 가리키는 포인터)
- ▶ count: 스택 항목의 개수(**선택사항**)

```
class LinkedStack :
    def __init__( self ):
        self.top = None
        self.count= 0
```

## 연결된 구조

### 2 연결된 스택의 구현

#### ● 기본 연산들

- ▶ 항목의 수

```
def size( self ):
    return self.count
```

- ▶ 스택 초기화

```
def clear( self ):
    self.top = None
    self.count = 0
```



# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결된 스택

#### 2 연결된 스택의 구현

##### ● 공백 상태와 포화 상태 검사

###### ▶ 공백 상태

```
def isEmpty( self ):
    return self.top == None
```

###### ▶ 포화 상태

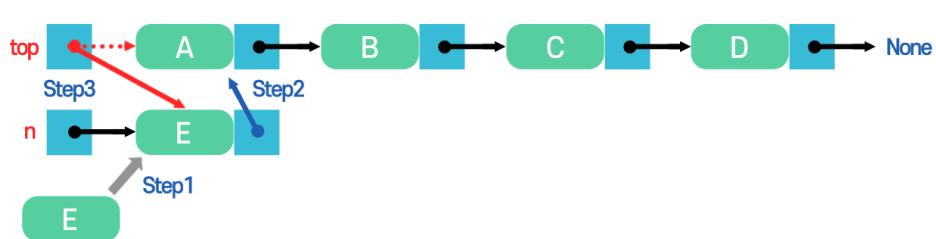
```
def isFull( self ):
    return False
```

## 연결된 구조

### 단순 연결 구조 응용 : 연결된 스택

#### 2 연결된 스택의 구현

##### ● 삽입 연산



- ① 입력 데이터 E를 이용해 새로운 노드 n을 생성함:  $n = \text{Node}(E)$
- ② N의 링크가 시작 노드를 가리키도록 함:  $n.link = \text{top}$
- ③ top이 n을 가리키도록 함:  $\text{top} = n$

# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결된 스택

#### 2 연결된 스택의 구현

##### ◎ 삽입 연산

```
def push( self, item ):
    n = Node(item, None)          # Step1
    n.link = self.top              # Step2
    self.top = n                   # Step3
    self.count += 1                # 스택 항목 수 증가
```

##### ▶ 동일한 코드

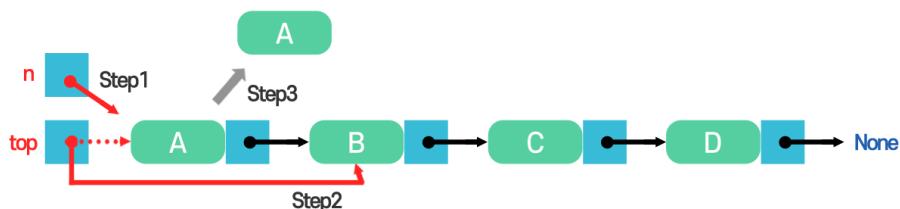
```
def push( self, item ):
    self.top = Node(item, self.top)  # 1~3 과정을 모두 포함
    self.count += 1                # 스택 항목 수 증가
```

## 연결된 구조

### 단순 연결 구조 응용 : 연결된 스택

#### 2 연결된 스택의 구현

##### ◎ 삭제 연산



- ① 변수 n이 시작 노드를 가리키도록 함: `n = top`
- ② top이 다음 노드를 가리키도록 함: `top = n.link`
- ③ n이 가리키는 노드의 데이터를 반환함: `return n.data`

# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결된 스택

#### 2 연결된 스택의 구현

##### ◎ 삭제 연산

- ▶ 메모리 해제를 신경 쓸 필요 없음
- ▶ 공백 검사 필요

```
def pop( self ):
    if not self.isEmpty():
        self.count -= 1
        data = self.top.data           # Step1
        self.top = self.top.link       # Step2
        return data                   # Step3
```

## 연결된 구조

### 단순 연결 구조 응용 : 연결된 스택

#### 2 연결된 스택의 구현

##### ◎ 기타 연산

peek( )

top의 data를 반환

```
def peek( self ):
    if not self.isEmpty():
        return self.top.data
```

# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결된 스택

## 2 연결된 스택의 구현

### ◎ 기타 연산

`display( )`

top부터 링크를 따라가면서 노드의 데이터를 출력

```
def display( self, msg='Stack: ' ):
    print(msg, end=' ')
    node = self.top
    while not node == None :
        print(node.data, end=' ')
        node = node.link
    print()
```

## 연결된 구조

### 단순 연결 구조 응용 : 연결된 스택

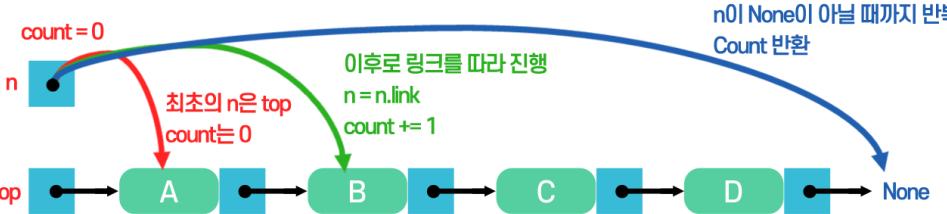
## 2 연결된 스택의 구현

### ◎ 전체 노드의 방문 예

`size( )`

count를 사용하지 않는 경우

- top부터 링크를 따라가면서 노드의 수를 계산함:  $O(n)$



# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 2 연결된 스택의 구현

#### ◎ 전체 노드의 방문 예

size( )

count를 사용하지 않는 경우

- top부터 링크를 따라가면서 노드의 수를 계산함: O(n)

```
def size1( self ):
    node = self.top          # top에서 부터
    cnt = 0                  # 항목의 수 초기화
    while not node == None : # 유효한 노드이면
        node = node.link    # 링크를 따라 움직임
        cnt += 1             # 항목의 수 증가
    return cnt               # 항목의 수 반환
```

# 단순 연결 구조 응용: 연결된 스택

The screenshot shows the PyCharm IDE interface with two files open: `Node.py` and `StackLinked.py`. The `Node.py` file contains a single class definition:`from Node import Node

class LinkedStack :
 def __init__( self ):
 self.top = None
 self.count= 0

 def size( self ):
 return self.count

 def clear( self ):
 self.top = None
 self.count = 0

 def isEmpty( self ):
 return self.top == None
 # return self.count == 0

 def isFull( self ):
 return False`

The `StackLinked.py` file is currently selected and shows its code:`from Node import Node

class LinkedStack :
 def __init__( self ):
 self.top = None
 self.count= 0

 def size( self ):
 return self.count

 def clear( self ):
 self.top = None
 self.count = 0

 def isEmpty( self ):
 return self.top == None
 # return self.count == 0

 def isFull( self ):
 return False`

The left sidebar displays a project tree with various Python files and their corresponding line numbers. The right sidebar shows the Python documentation for the `clear` method.

실습 단계
Node 클래스 import
스택 클래스는 top과 count 변수 사용
push 연산에서는 count +1, pop 연산에서는 count -1
push 연산은 포화 검사 X, pop 연산은 공백 검사 O
짝수는 even에 저장, 홀수는 odd에 저장



# 단순 연결 구조 응용: 연결 리스트

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

#### 1 단순 연결 리스트의 구조

##### ● 리스트의 추상 자료형

데이터	연산
같은 유형의 서로 비교할 수 있는 요소들	<ul style="list-style-type: none"> <li>▪ <code>List( )</code>: 비어 있는 새로운 리스트를 만듦</li> <li>▪ <code>Insert(pos, e)</code>: pos 위치에 새로운 요소 e를 삽입</li> <li>▪ <code>delete(pos)</code>: pos 위치에 있는 요소를 꺼내고(삭제) 반환</li> <li>▪ <code>isEmpty( )</code>: 리스트가 비어 있는지를 검사</li> <li>▪ <code>isFull( )</code>: 리스트가 가득 차 있는지를 검사</li> <li>▪ <code>getEntry(pos)</code>: pos 위치에 있는 요소를 반환</li> <li>▪ <code>size( )</code>: 리스트 안의 요소의 개수를 반환</li> </ul>

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

#### 1 단순 연결 리스트의 구조

##### ● 리스트의 추상 자료형

데이터	연산
같은 유형의 서로 비교할 수 있는 요소들	<ul style="list-style-type: none"> <li>▪ <code>clear( )</code>: 리스트를 초기화</li> <li>▪ <code>find(e)</code>: 리스트에서 항목 e가 있는지 찾아 인덱스를 반환</li> <li>▪ <code>replace(pos, e)</code>: pos에 있는 항목을 e로 바꿈</li> <li>▪ <code>append( )</code>: 리스트의 맨 뒤에 새로운 항목을 추가</li> <li>▪ <code>pop( )</code>: 리스트의 맨 뒤 항목을 꺼내고 반환</li> <li>▪ <code>display( )</code>: 리스트를 화면에 보기 좋게 출력</li> </ul>

# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

#### 1 단순 연결 리스트의 구조

##### 단순 연결된 리스트 구조



## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

#### 2 단순 연결 리스트 구현

##### ● 클래스 정의와 멤버 변수

▶ head: 시작 노드(인덱스 0)를 가리키는 포인터

▶ count: 리스트 항목의 개수(**선택사항**)

```

class LinkedList:
    def __init__( self ):
        self.head = None
        self.count = 0

    def size( self ) : return self.count
    def isEmpty( self ): return self.head == None
    def clear( self ) :
        self.head = None
        self.count = 0
  
```

# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

#### 2 단순 연결 리스트 구현

##### ◎ 항목 접근 연산

▶ 인덱스 pos를 이용한 노드 접근:  $O(n)$

```
def getNode(self, pos) :
    if pos < 0 : return None          # 인덱스가 pos인 노드 반환
    if pos >= self.size() :           # None 반환
        pos = self.size()-1           # 맨 뒤의 노드 반환
    node = self.head
    while pos > 0 and node != None :
        node = node.link
        pos -= 1
    return node
```

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

#### 2 단순 연결 리스트 구현

##### ◎ 항목 접근 연산

▶ 인덱스 pos를 이용한 항목(데이터) 접근:  $O(n)$

```
def getEntry(self, pos) :
    node = self.getNode(pos)          # pos번째의 항목 반환
    if node == None :                 # pos번째의 노드
        return None
    else :                           # 노드의 항목만 반환
        return node.data
```

# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

#### 2 단순 연결 리스트 구현

##### ● 전체 노드의 방문

▶ size\_iter( ), display( ), find( ) 등

```
def size_iter( self ) : # count 미 사용시
    node = self.head;
    count = 0
    while node is not None :
        node = node.link
        count += 1
    return count
```

```
def display(self, msg='LinkedList:'):
    print(msg, end='')
    node = self.head
    while node is not None :
        print(node.data, end='->')
        node = node.link
    print('None')
```

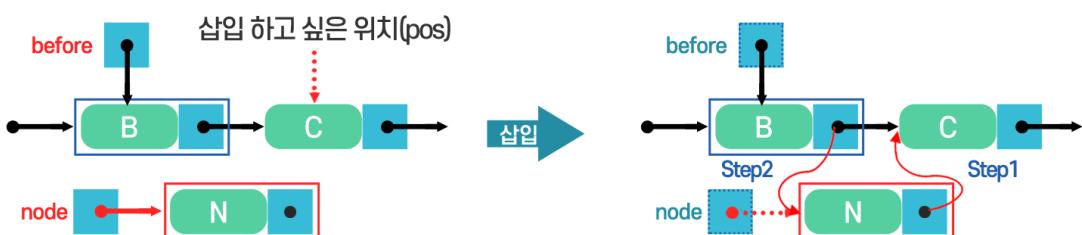
## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

#### 2 단순 연결 리스트 구현

##### ● 삽입 연산

▶ insert(pos, elem)



① 노드 N이 노드 C를 가리키게 함: node.link = before.link

② 노드 B가 노드 N을 가리키게 함: before.link = node

# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

## 2 단순 연결 리스트 구현

### ● 삽입 연산

▶ `insert(pos, elem)`

```
def insert(self, pos, elem) :
    before = self.getNode(pos-1)           # before 노드를 찾음
    if before == None :                   # 맨 앞에 삽입하는 경우
        self.head = Node(elem, self.head)
    else :                               # 중간에 삽입하는 경우
        node = Node(elem, before.link)     # 노드 생성 + Step1
        before.link = node                # Step2
    self.count += 1
```

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

## 2 단순 연결 리스트 구현

### ● 삭제 연산

▶ `delete(pos)`



- 1 before의 link가 삭제할 노드의 다음 노드를 가리키도록 함: `before.link = before.link.link`

# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

## 2 단순 연결 리스트 구현

### ● 삭제 연산

▶ delete(pos)

```
def delete(self, pos) :
    before = self.getNode(pos-1)
    if before == None :
        if self.head is not None :
            self.head = self.head.link
    else:
        before.link = before.link.link
    self.count -= 1
```

# before 노드를 찾음  
# 맨 앞 노드를 삭제하는 경우  
# 중간 노드를 삭제하는 경우  
# Step1

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

## 2 단순 연결 리스트 구현

### ● 테스트 프로그램

```
if __name__ == "__main__":
    s = LinkedList()
    print('<연결된 구조로 구현한 리스트(LinkedList) 테스트>')
    s.display("LinkedList( 최초 ): ")
    s.insert(0, 10)
    s.insert(0, 20)
    s.insert(1, 30)
    s.insert(s.size(), 40)
    s.insert(2, 50)
    s.display("LinkedList(삽입x5): ")
    s.replace(2, 90)
    s.display("LinkedList(교체x1): ")
    s.delete(2)
```



# 단순 연결 구조 응용: 연결된 스택

## 연결된 구조

### 단순 연결 구조 응용 : 연결 리스트

#### 2 단순 연결 리스트 구현

##### ◎ 테스트 프로그램

```
s.delete(s.size() - 1)
s.delete(0)
s.display("LinkedList(삭제x3): ")
s.clear()
s.display("LinkedList(정리후): ")
```

```
C:\WINDOWS\system32\cmd.exe
<연결된 구조로 구현한 리스트(LinkedList) 테스트>
LinkedList(최초): None
LinkedList(삽입x5): 20->30->50->10->40->None
LinkedList(교체x1): 20->30->90->10->40->None
LinkedList(삭제x3): 30->10->None
LinkedList(정리후): None
```