

파이썬 데이터처리 및 분석

데이터 구조

2020.07. 15. 수요일

최희련

En-CORE

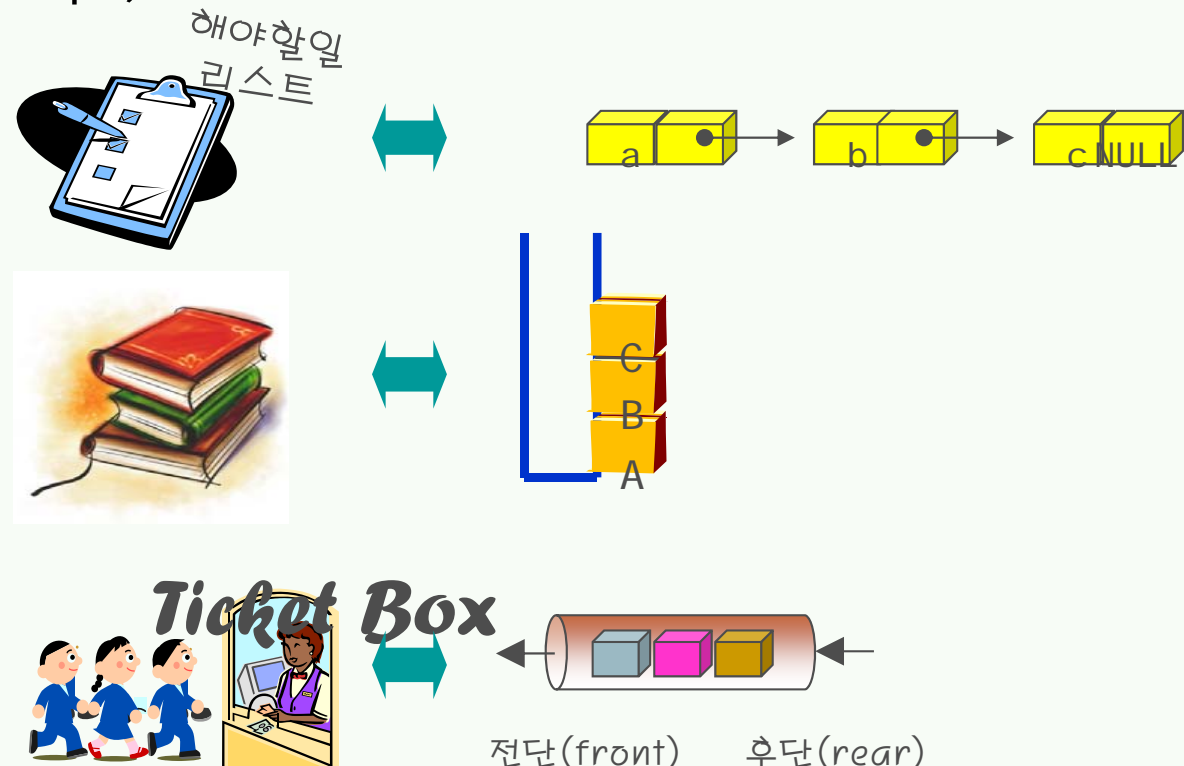
Data Science Edu.

일상생활과 자료구조의 관계

- 자료구조: 데이터와 그들의 관계를 조직화 구조화 한 것, 데이터를 효율적으로 조직, 저장하는 방법

- 선형형태: 리스트(List), 스택(Stack), 큐(Queue)
- 비선형형태: 트리(Tree), 그래프(Graph)

일상생활에서의 예	자료구조
물건을 쌓아두는 것	스택
영화관 매표소의 줄	큐
할일 리스트	리스트
지도	그래프
조직도	트리



그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

알고리즘(Algorithm) 정의

■ 알고리즘

○ 주어진 문제를 해결하기 위한 절차 또는 단계들의 유한한 집합

→ 잘 정의된 문제 해결 과정

○ 장치(컴퓨터 등)가 이해할 수 있는 언어로 기술되어야 함

○ 알고리즘의 조건

- 입력: 0개 이상
- 출력: 1개 이상
- 명백성: 각 명령어의 의미가 명확
- 유한성: 한정된 수의 단계 후에는 반드시 종료
- 유효성: 각 명령어들은 실행 가능한 연산

** 알고리즘의 기술 방법

- 자연어
- 흐름도(flowchart)
- 유사코드(pseudo-code)
- 프로그래밍 언어

<유사코드>

ArrayMax(A,n)

```
tmp ← A[0];  
for i ← 1 to n-1 do  
    if tmp < A[i] then  
        tmp ← A[i];  
return tmp;
```

알고리즘의 성능분석

■ 효율적 알고리즘

- 알고리즘의 시작 → 결과가 나올 때까지 실행 시간이 짧고, 컴퓨터 메모리 같은 자원을 적게 사용하는 알고리즘

■ 실행 시간 측정방법

- 알고리즘을 구현하여 시작과 끝의 간격을 시간함수를 사용하여 측정

■ 알고리즘의 복잡도 분석방법

- 시간 복잡도: 알고리즘을 이루는 연산들의 실행 수
- 시간 복잡도 함수: 연산의 개수를 입력의 개수 n 의 함수로 나타낸 것 $T(n)$

시간 복잡도 분석

■ 양의 정수 n 을 n 번 더하는 문제

알고리즘 A	알고리즘 B	알고리즘 C
$\text{sum} \leftarrow n * n;$	$\text{sum} \leftarrow 0;$ for $i \leftarrow 1$ to n do $\text{sum} \leftarrow \text{sum} + n;$	$\text{sum} \leftarrow 0;$ for $i \leftarrow 1$ to n do for $j \leftarrow 1$ to n do $\text{sum} \leftarrow \text{sum} + 1;$

	알고리즘 A	알고리즘 B	알고리즘 C
대입연산	1	$n + 1$	$n * n + 1$
덧셈연산		n	$n * n$
곱셈연산	1		
나눗셈연산			
전체연산수	2	$2n + 1$	$2n^2 + 1$

알고리즘의 복잡도 표현 방법(점근적 표기)

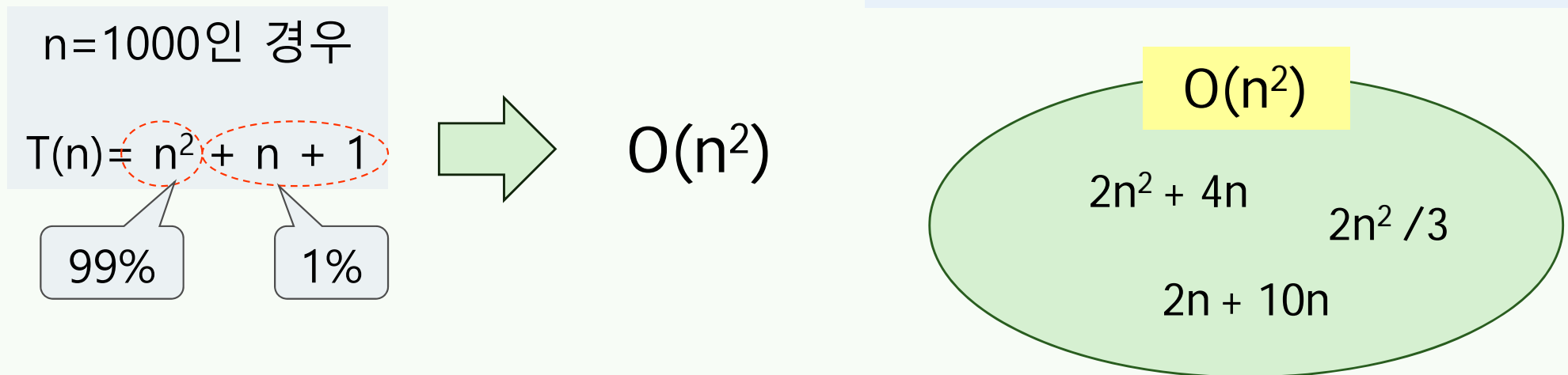
■ 빅오(Big-oh notation) : 최악인 경우의 알고리즘 수행 시간을 나타냄

- 충분히 큰 입력값에 대해 차수가 가장 큰 항이 가장 영향을 크게 미치고 다른 항들은 상대적으로 무시될 수 있음, 점근적 상한

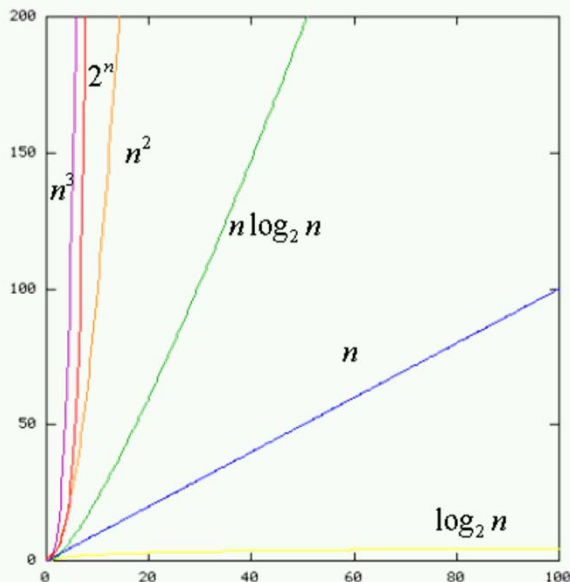
(예: 기껏해야 내 키는 8미터 이하입니다)

- 시간 복잡도의 증가 함수에서 불필요한 정보를 제거한 방법

어떤 알고리즘을 수행하는 증가함수의 종류가 다음과 같다면?



- $O(1)$: 상수형
- $O(\log_2 n)$: 로그형
- $O(n)$: 선형
- $O(n \log_2 n)$: 로그선형
- $O(n^2)$: 2차형
- $O(n^3)$: 3차형
- $O(n^k)$: k차형
- $O(2^n)$: 지수형
- $O(n!)$: 팩토리얼형

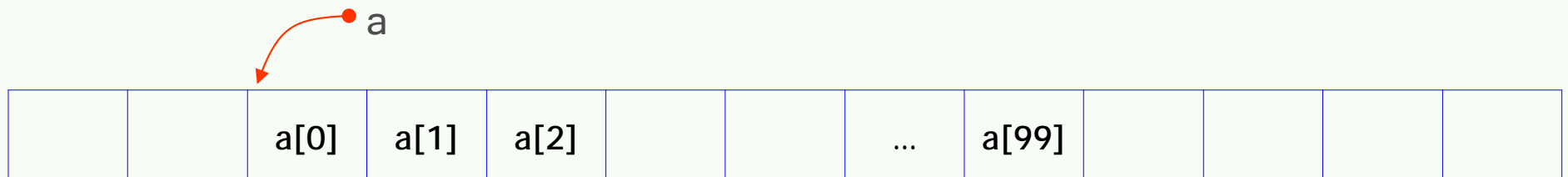


시간복잡도	n					
	1	2	4	8	16	32
1	1	1	1	1	1	1
$\log n$	0	1	2	3	4	5
n	1	2	4	8	16	32
$n \log n$	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
$n!$	1	2	24	40320	20922789888000	26313×10^{33}

자료형 기초 개념 - 배열

■ 배열

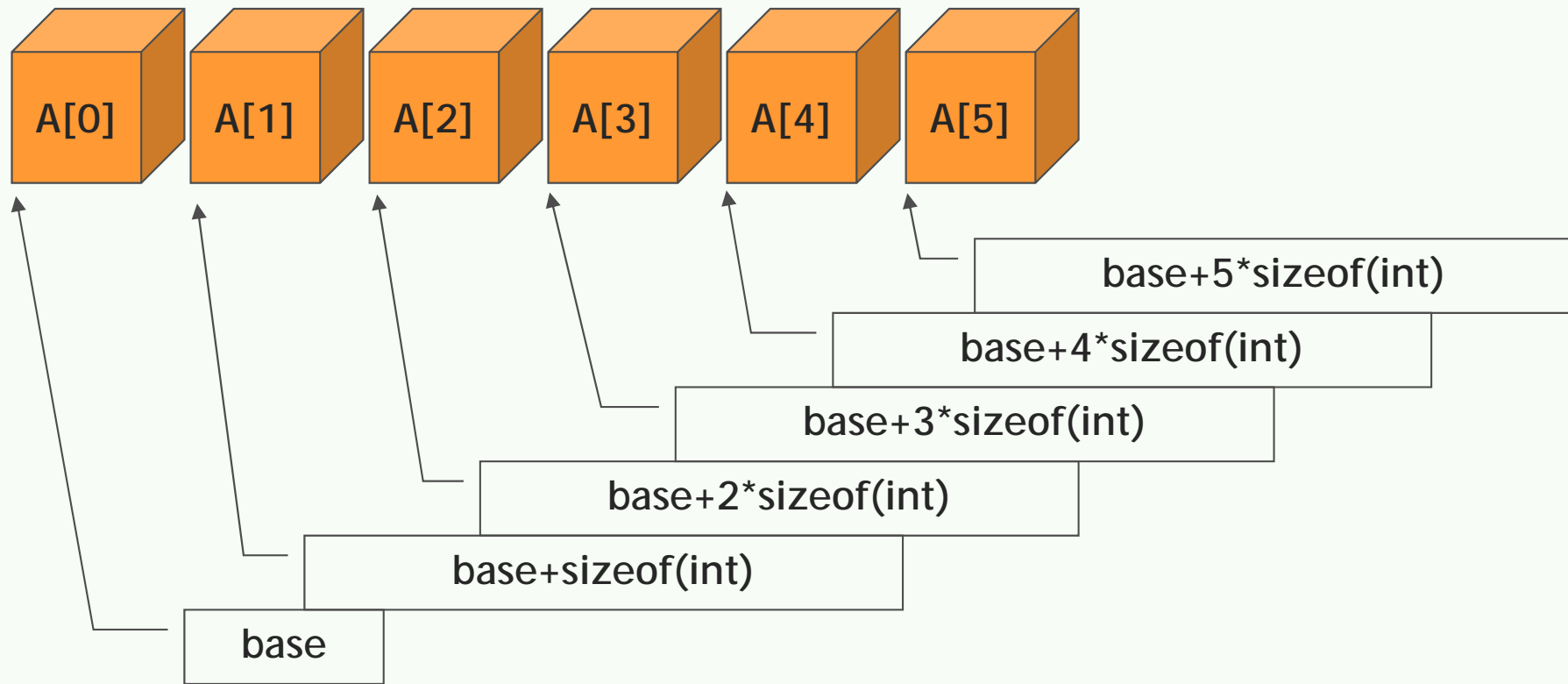
- 같은 데이터형을 가지는 변수들의 집합
- 구조



- 각 배열의 원소는 []안에 첨자(index)를 이용하여 참조
- 배열 원소의 시작번호는 0 임
- 배열의 각 원소는 메모리에 연속적으로 저장
- 배열의 이름은 메모리상에서 인덱스번호 0을 가지는 원소의 주소(위치)를 나타냄

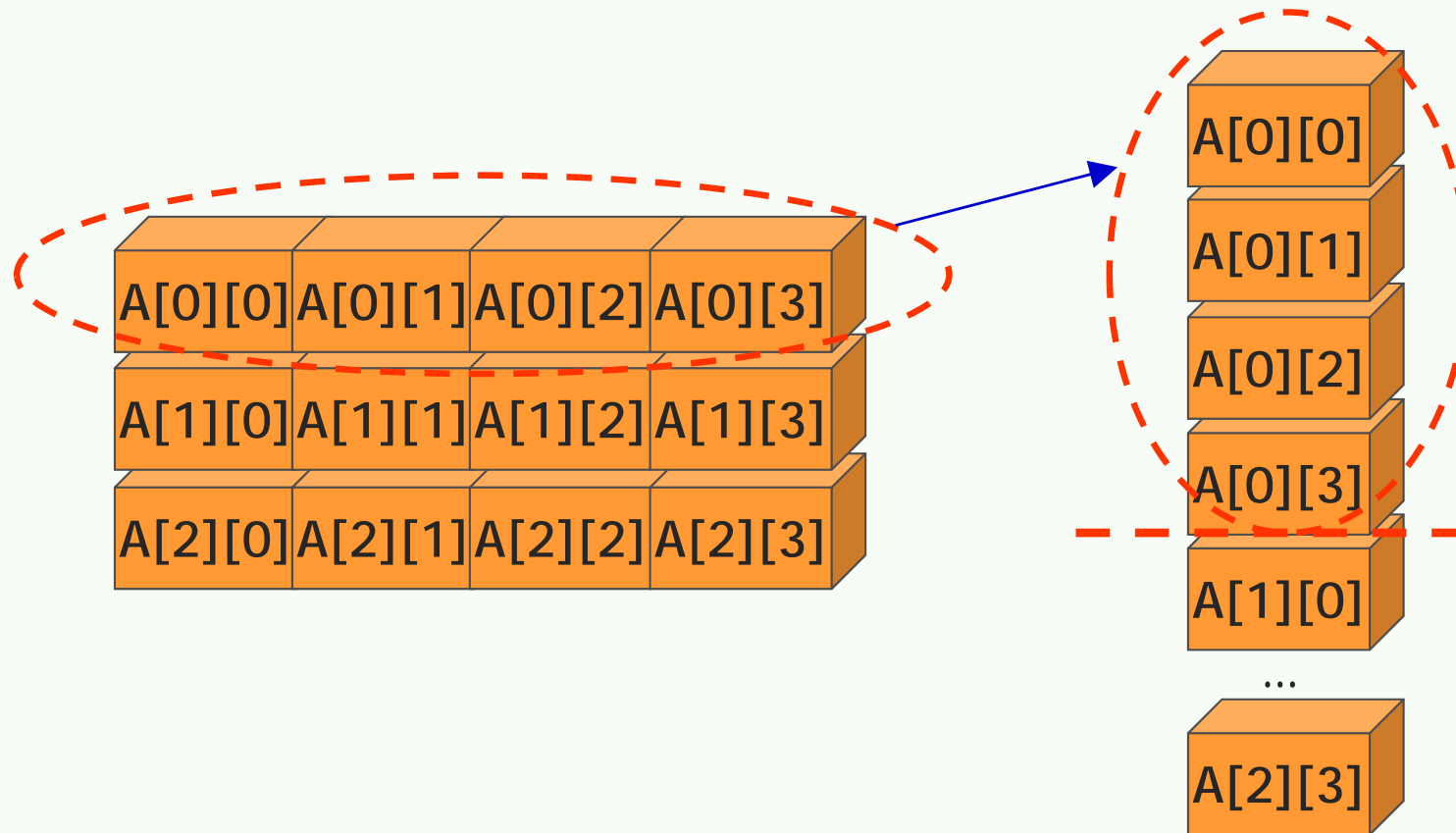
1차원 배열

■ [] 한 개로 구성



2차원 배열

- [][] 의 두 개로 구성
- 첫번째 [] 행(row) 인덱스를, 두번째 []은 열(column) 인덱스를 나타냄



실제 메모리안에서의 위치

그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

List

■ 자료를 순서대로 저장하는 자료구조, 즉 순서를 가진 항목들의 모임

○ 일직선으로 연결된 자료



■ 리스트의 예

- 요일(월, 화, 수, ...), 파일의 목록, 카드(Ace, 2, 3, ... , King)
- 한글 자음의 모임

그림참조: 열혈강의-C로만드는 자료구조와 적용알고리즘

List의 추상 자료형(ADT)

·객체: n개의 element형으로 구성된 순서있는 모임

·연산:

- `add_last(list, item) ::=` 맨끝에 요소를 추가한다.
- `add_first(list, item) ::=` 맨앞에 요소를 추가한다.
- `add(list, pos, item) ::=` pos 위치에 요소를 추가한다.
- `deleteF(list, pos) ::=` pos 위치의 요소를 제거한다.
- `clear(list) ::=` 리스트의 모든 요소를 제거한다.
- `replace(list, pos, item) ::=` pos 위치의 요소를 item로 바꾼다.
- `is_in_list(list, item) ::=` item이 리스트안에 있는지를 검사한다.
- `get_entry(list, pos) ::=` pos 위치의 요소를 반환한다.
- `get_length(list) ::=` 리스트의 길이를 구한다.
- `is_empty(list) ::=` 리스트가 비었는지를 검사한다.
- `is_full(list) ::=` 리스트가 꽉찼는지를 검사한다.
- `display(list) ::=` 리스트의 모든 요소를 표시한다.

List 구현 방법

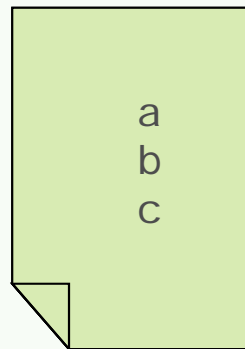
■ 배열을 이용하는 방법

- 구현이 간단
- 삽입, 삭제시 오버헤드
- 항목의 개수 제한

■ 연결리스트를 이용하는 방법

- 배열에 비해 구현이 복잡
- 삽입, 삭제가 효율적
- 크기가 제한되지 않음

리스트 ADT



배열을 이용한 구현



연결리스트를 이용한 구현



그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

배열 구조 - 생성(1)

■ 배열 구조 이용 시

○ 정적 생성:

1. 생성할 데이터 사이즈에 맞춰 배열 생성
MAX_SIZE = n
2. 데이터 자료형 결정
정수, 실수, 문자열 등
3. 구현할 언어의 문법에 맞춰 코드 작성

○ 예시

```
//C언어
typedef struct{
    int list[MAX_SIZE];
    int size;
}ArrayList

void init(ArrayList *AL){
    AL->size = 0 //(*AL).size = 0
}
```

```
#파이썬
def array_init():
    ArrayList = []
```

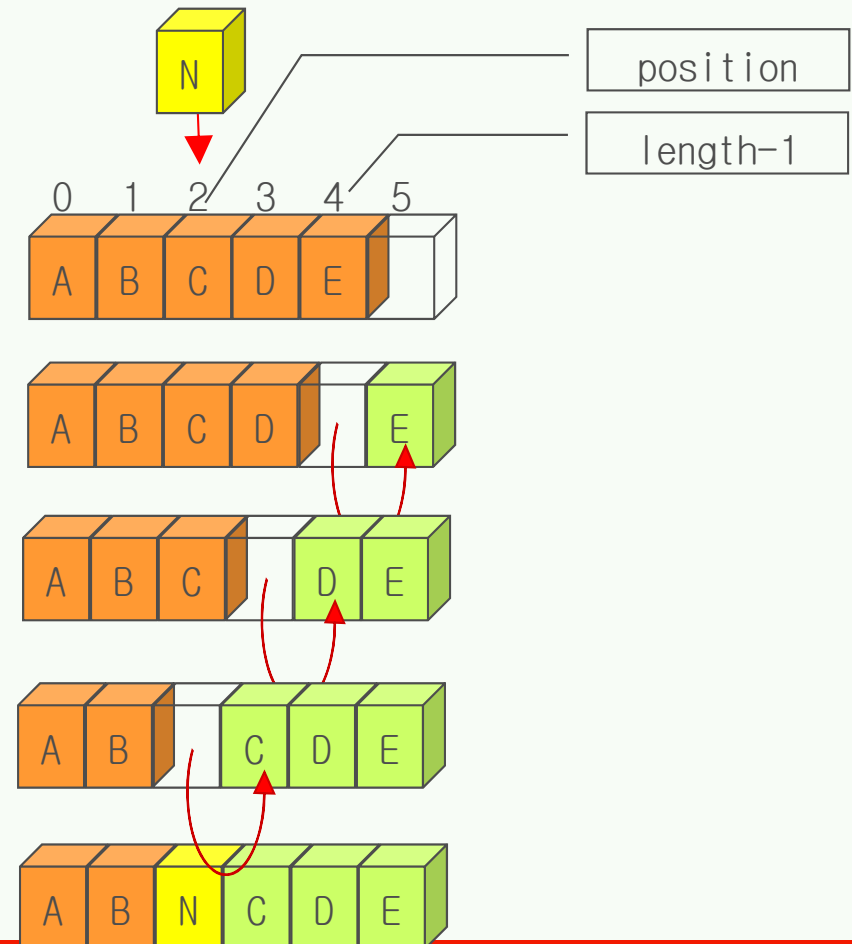
배열구조 - 추가

■ 추가 또는 삽입(insert) 연산

- 배열의 포화상태 검사, 즉 데이터 추가 가능성 판단
- 추가 가능하다 판단되면, 추가 또는 삽입 위치 다음에 있는 자료들을 한 칸씩 뒤로 이동

//ArrayList: 생성된 배열리스트, item:추가할 데이터,
//p: 추가할 위치

```
add 또는 insert function(ArrayList, item, p){  
    if ArrayList is empty:  
        Error 또는 ArrayList[0] = item  
    else if ArrayList isn't full and p >= 0 and p  
        < ArrayList의 입력 size(즉, 입력갯수):  
        for 입력 size - 1 에서부터 p위치까지 -1감소:  
            ArrayList[i+1] = ArrayList[i]  
        ArrayList[p] = item  
        size +1 증가  
}
```



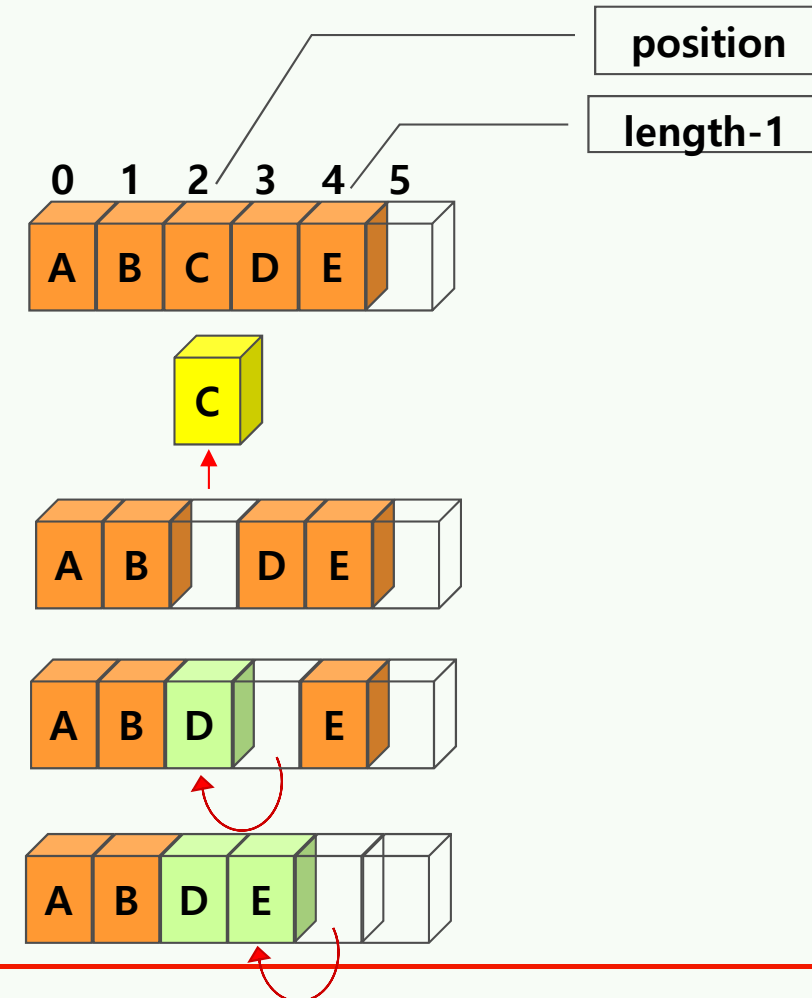
그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사
En-CORE, Data Science 교육, 최희련

배열구조 - 삭제(delete)

■ 삭제 연산

- 삭제할 데이터의 유무 확인, empty 여부
- 삭제할 위치 확인 후, 삭제할 위치 데이터부터 마지막 데이터들을 왼쪽으로 이동

```
//ArrayList: 삭제대상의 배열리스트
//p: 삭제위치
//반환값: 삭제되는 데이터
delete function(ArrayList, p){
    if ArrayList is empty and  $p < 0$  or  $p \geq$  ArrayList size:
        Error
    else:
        item = ArrayList[p]
        for p에서 부터 ArrayList size -1 까지 1증가:
            ArrayList[i] = ArrayList[i+1]
        ArrayList[p] = item
        size 1 감소
    return item
}
```

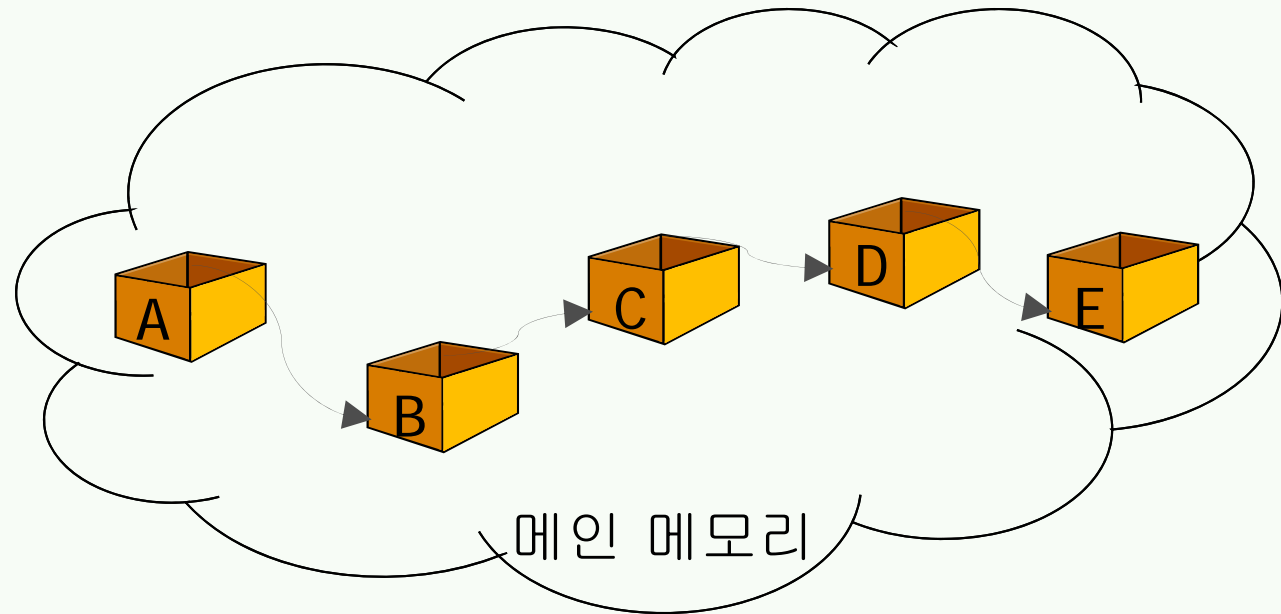


연결리스트(Linked List)

- 리스트의 항목들을 노드(node)라는 곳에 분산하여 저장
- 항목 + 다음 항목을 가리키는 주소 로 구성
- 노드(node): <항목, 주소>, 항목 필드(데이터) + 주소 필드(포인터)
- 노드의 물리적 순서와 논리적 순서가 일치하지 않을 수 있음

■ 연결리스트 종류

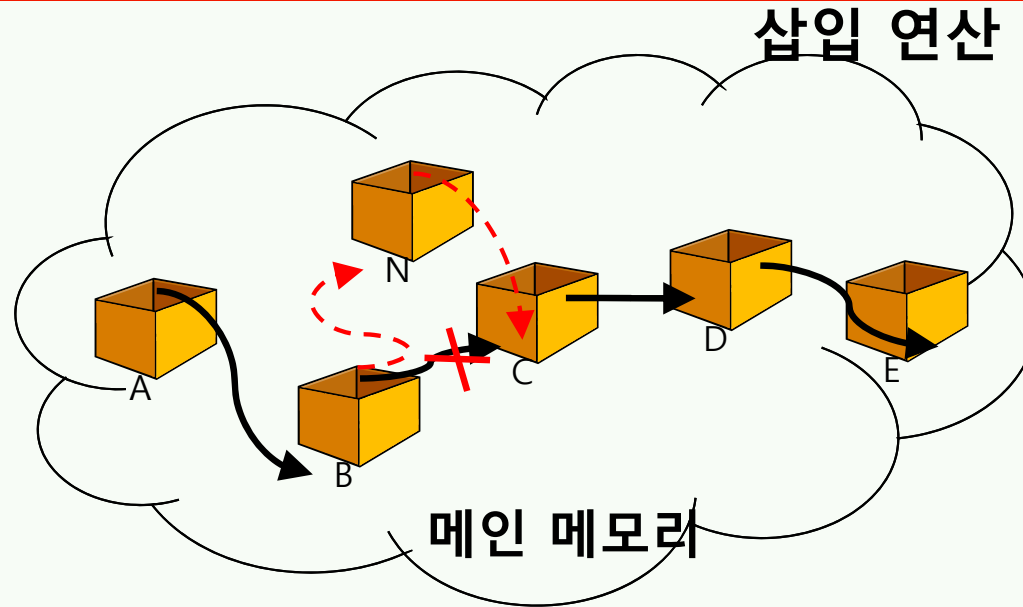
- 단순연결리스트(Single)
- 이중연결리스트(Double)
- 원형연결리스트(Circle)



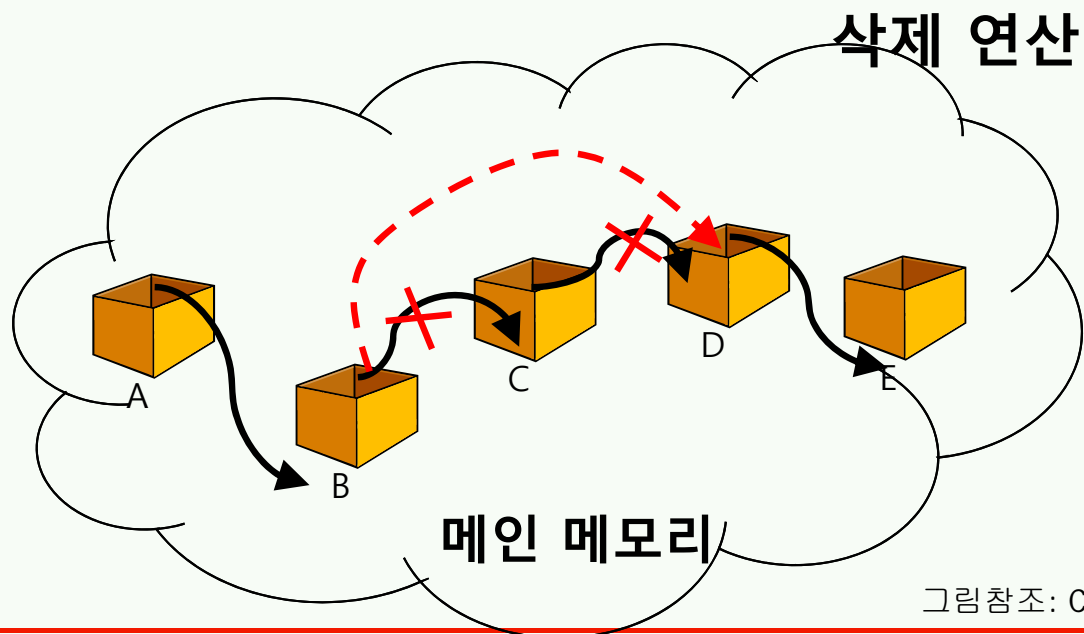
그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

연결리스트의 삽입/삭제 개념

■ 삽입



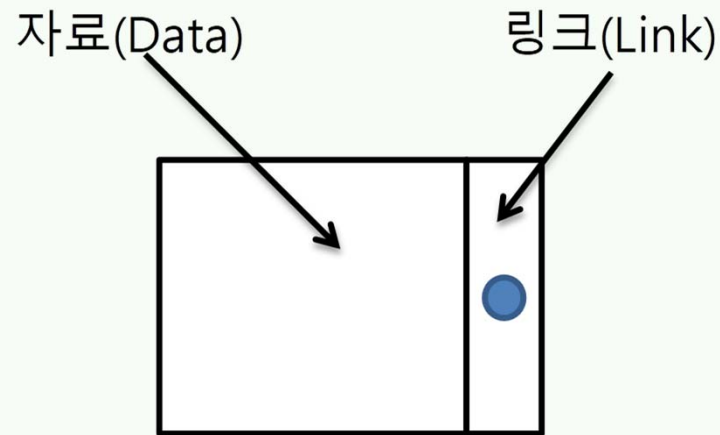
■ 삭제



그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

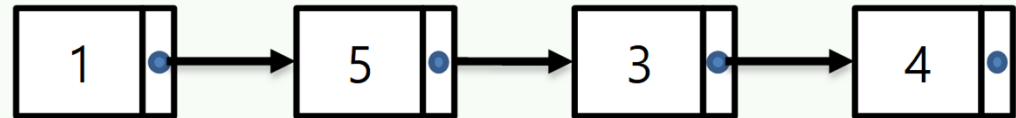
연결리스트 구조(1)

■ 노드 (node) ?

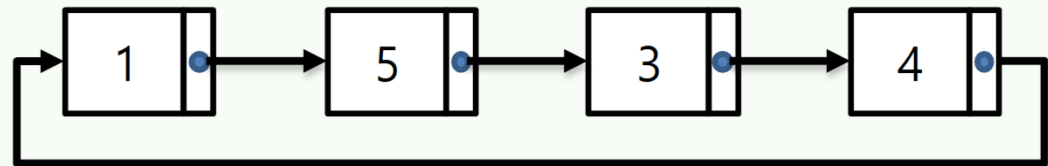


■ 연결 리스트의 종류

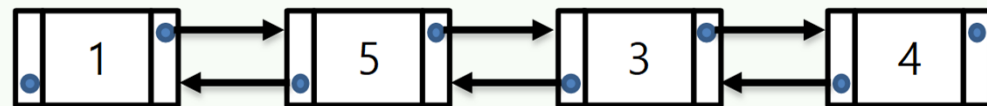
○ 단순 연결(Singly)



○ 원형 연결(Circular)

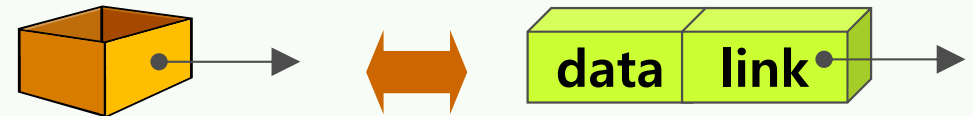


○ 이중 연결(Double)

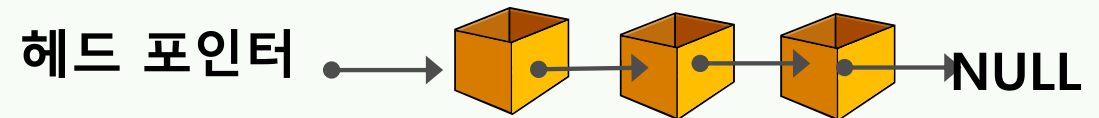


연결리스트 구조(2)

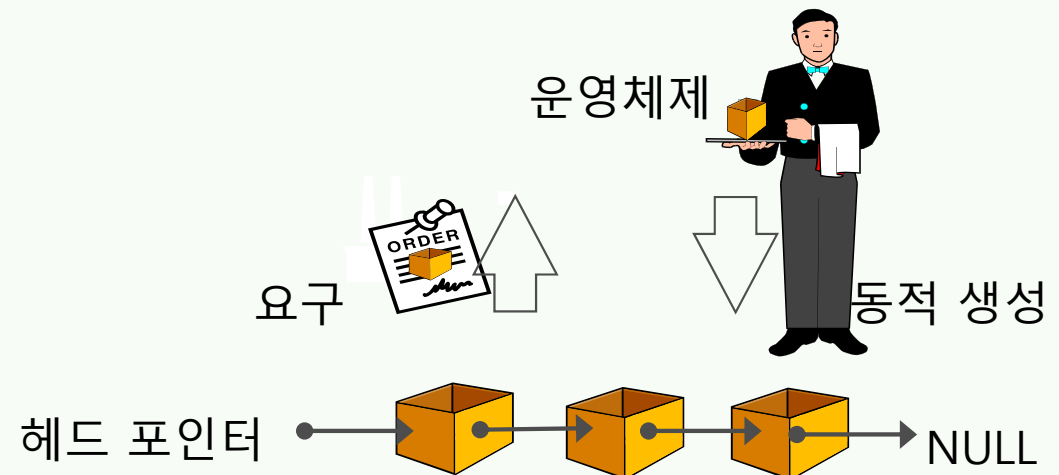
- 노드 = 데이터 필드 + 링크 필드



- 헤드 포인터(head pointer):
리스트의 첫번째 노드를 가리키는
변수



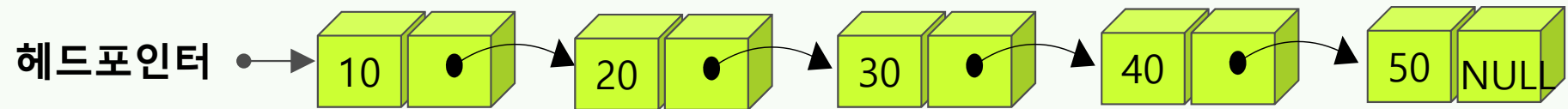
- 노드의 생성: 필요할 때마다
동적 메모리 생성 이용하여
노드를 생성



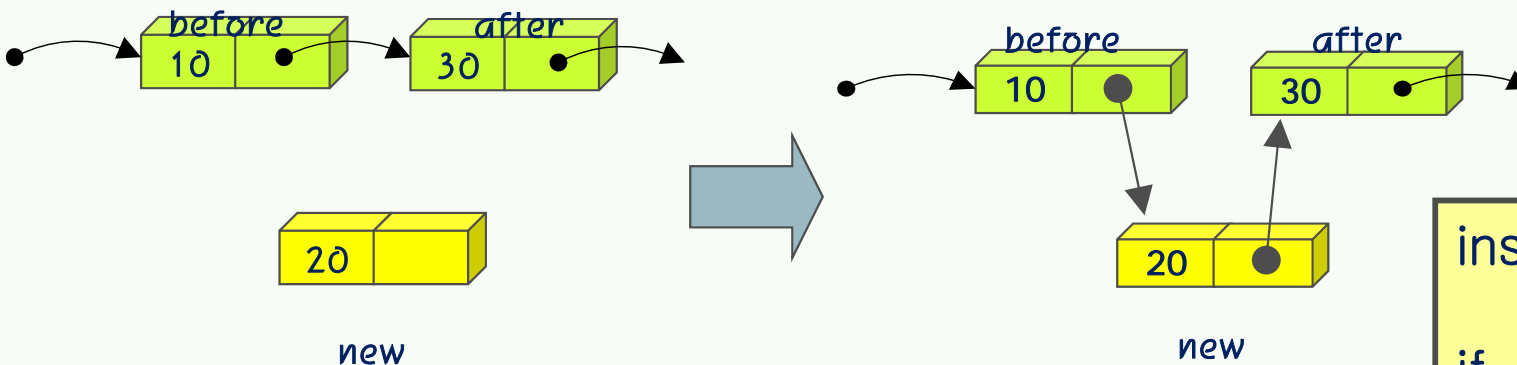
그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

단순연결리스트(Single)

- 하나의 링크 필드를 이용하여 연결
- 마지막 노드의 링크값은 NULL



- 삽입연산

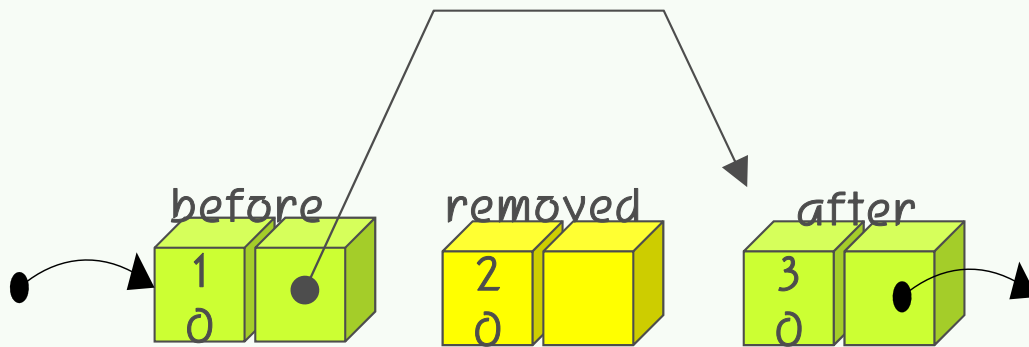
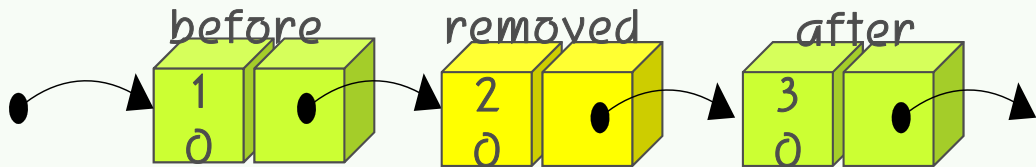


```
insert_node(L, before, new)
```

```
if L = NULL  
then L ← new  
else new.link ← before.link  
before.link ← new
```

단순연결리스트(Single)

■ 삭제연산



```
remove_node(L, before, removed)
```

```
if L ≠ NULL
```

```
then
```

```
    before.link ← removed.link
```

```
    destroy(removed)
```

연결리스트 구현 코드

■ 파이썬 코드

```
class Node:
    size = 0
    def __init__(self, item, link):
        self.item = item
        self.next = link
        Node.size += 1

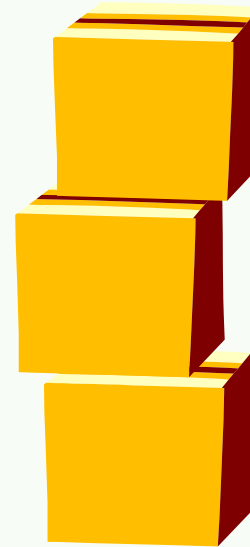
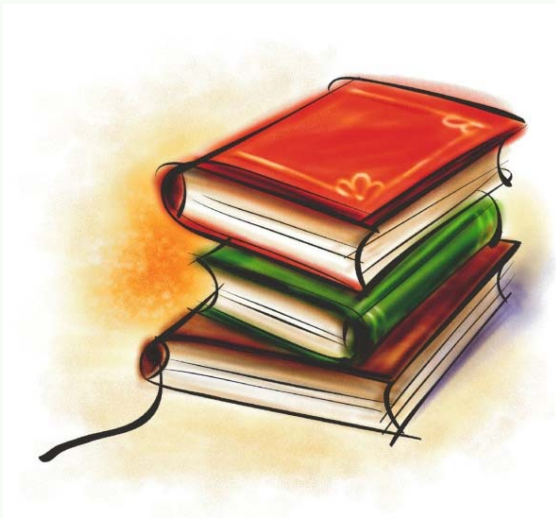
def insert_front(item):
    global head
    if Node.size != 0:
        head = Node(item, head)
    else:
        head = Node(item, None)

def print_list():
    p = head
    while p:
        if p.next != None:
            print(f'{p.item} --> ', end="")
        else:
            print(p.item)
        p = p.next
```

```
if __name__ == '__main__':
    insert_front('apple')
    insert_front('cherry')
    insert_front('banana')
    print_list()
```

스택이란?

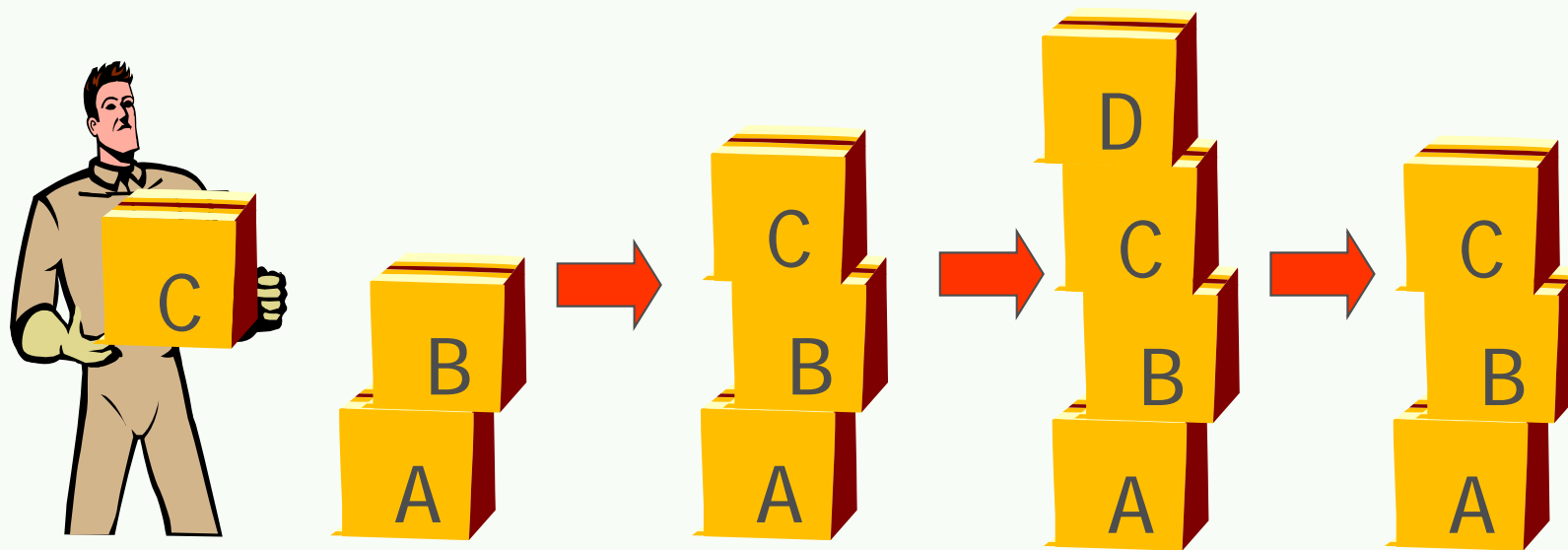
- 스택(stack): 쌓아놓은 더미



그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

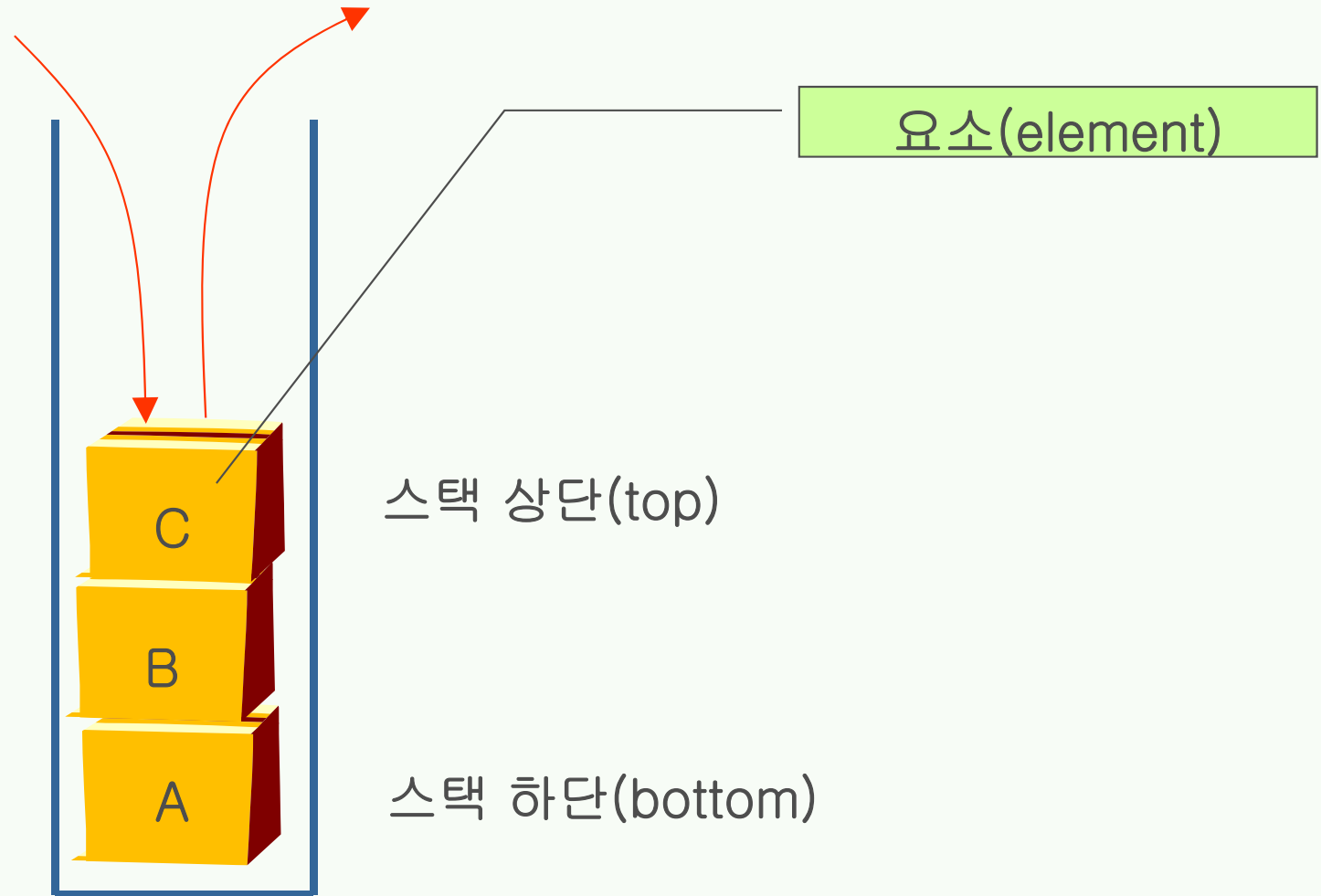
스택의 특징

- 후입선출(LIFO: Last-In First-Out, FILO: First-In Last-Out): 가장 최근에 들어온 데이터가 가장 먼저 나감.



그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

스택의 구조



그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

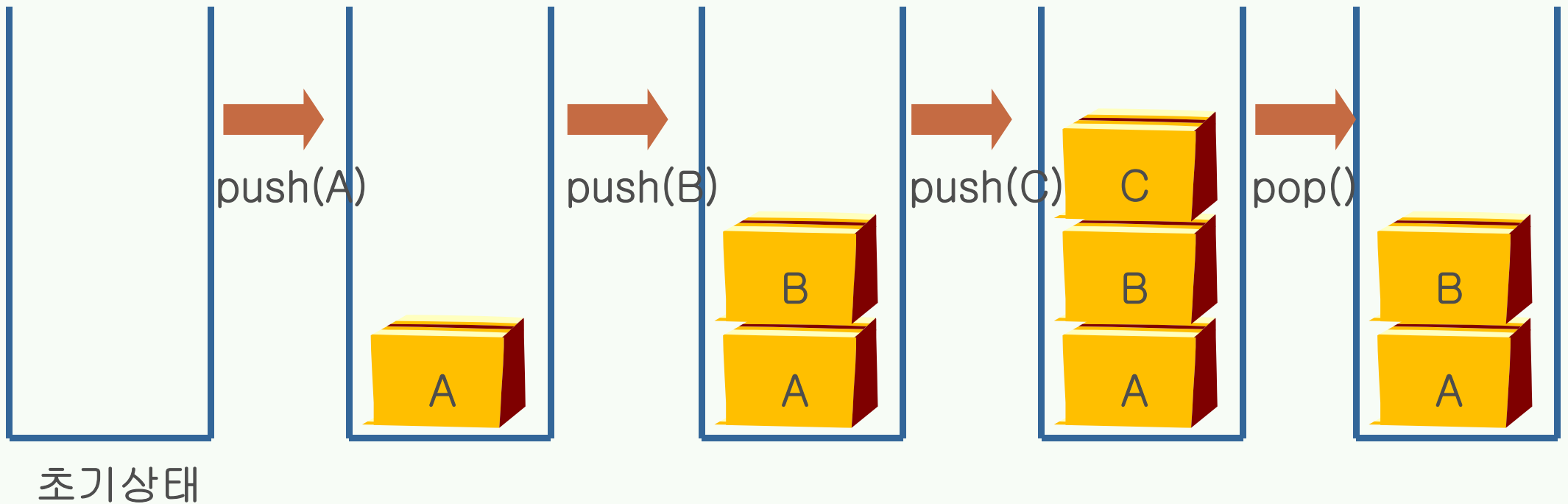
스택 추상데이터타입(ADT)

- 객체: n 개의 element형의 요소들의 선형 리스트
- 연산:
 - $\text{create}() ::=$ 스택을 생성한다.
 - $\text{is_empty}(s) ::=$ 스택이 비어있는지를 검사한다.
 - $\text{is_full}(s) ::=$ 스택이 가득 찼는가를 검사한다.
 - $\text{push}(s, e) ::=$ 스택의 맨 위에 요소 e 를 추가한다.
 - $\text{pop}(s) ::=$ 스택의 맨 위에 있는 요소를 삭제한다.
 - $\text{peek}(s) ::=$ 스택의 맨 위에 있는 요소를 삭제하지 않고 반환한다.

- $\text{create}() ::$ n 개의 배열 또는 n 개의 링크드리스트 형태로 스택 생성
- $\text{push} ::$ 배열의 형태일 경우는 요소를 더 추가할 수 있는 공간이 있는지 판단 여부가 필요
- $\text{pop} ::$ 배열 또는 링크드리스트 형태 둘 다 공백스택에 대한 판단 여부 필요

스택의 연산

■ 삽입(push), 삭제(pop)

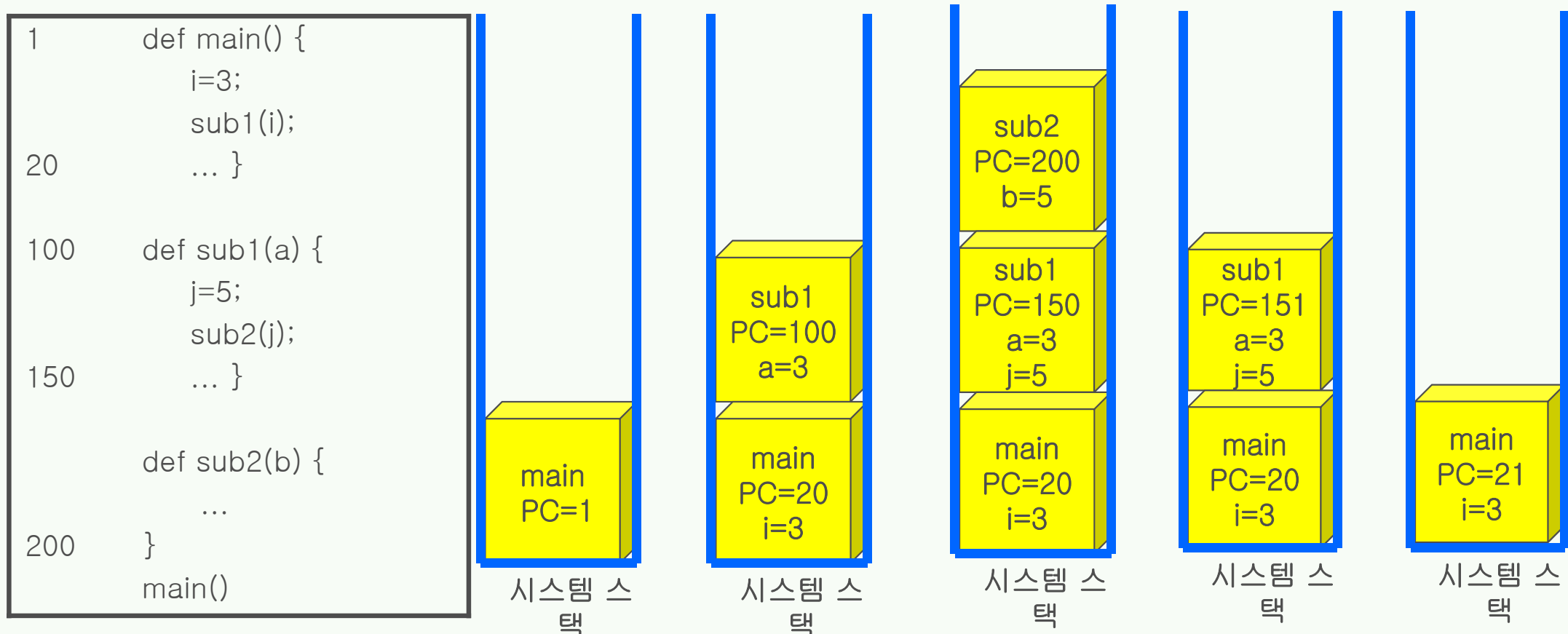


그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

스택의 용도

■ 입력과 역순의 출력이 필요한 경우

- 에디터에서 되돌리기(undo) 기능
- 함수호출에서 복귀주소 기억



파이썬 구현

```
def push(stack,item):
    stack.append(item)

def pop(stack):
    if len(stack) != 0:
        item = stack.pop(-1)# stack.pop()
        return item
    else:
        empty()

def empty():
    print("Stack is Empty")

def top(stack):
    if len(stack) != 0:
        return stack[-1]
    else:
        empty()
```

```
if __name__ == '__main__':
    me = []
    pop(me)
    push(me,'apple')
    push(me,'orange')
    push(me,'cherry')
    print(me)
    print(top(me))
    pop(me)
    print(me)
```

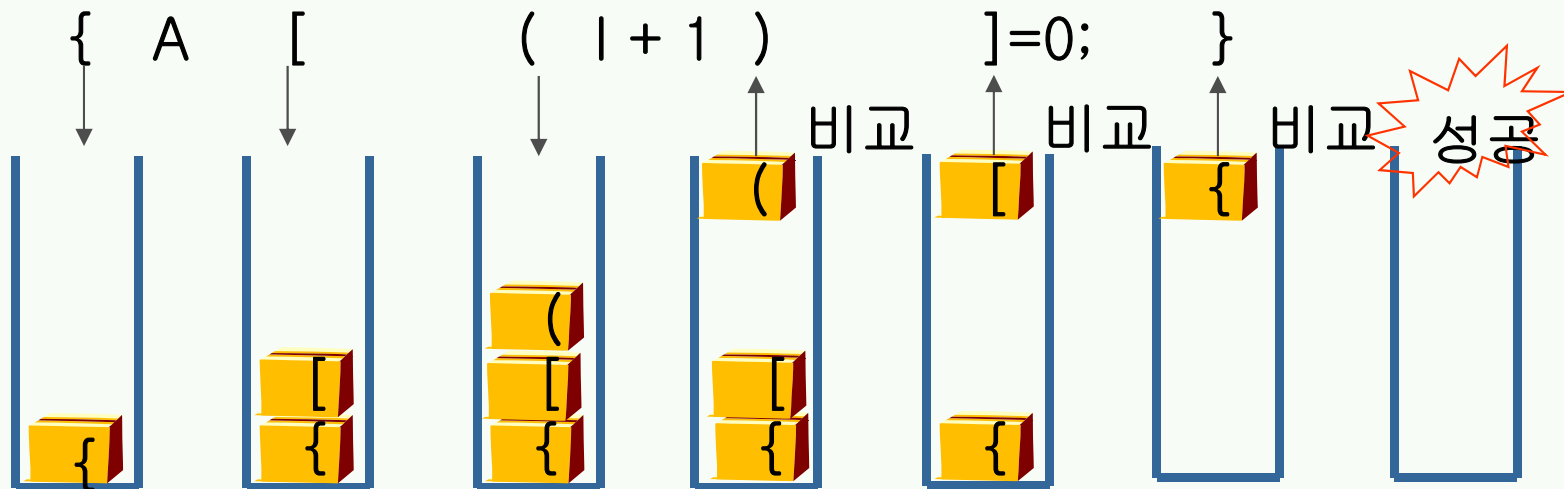
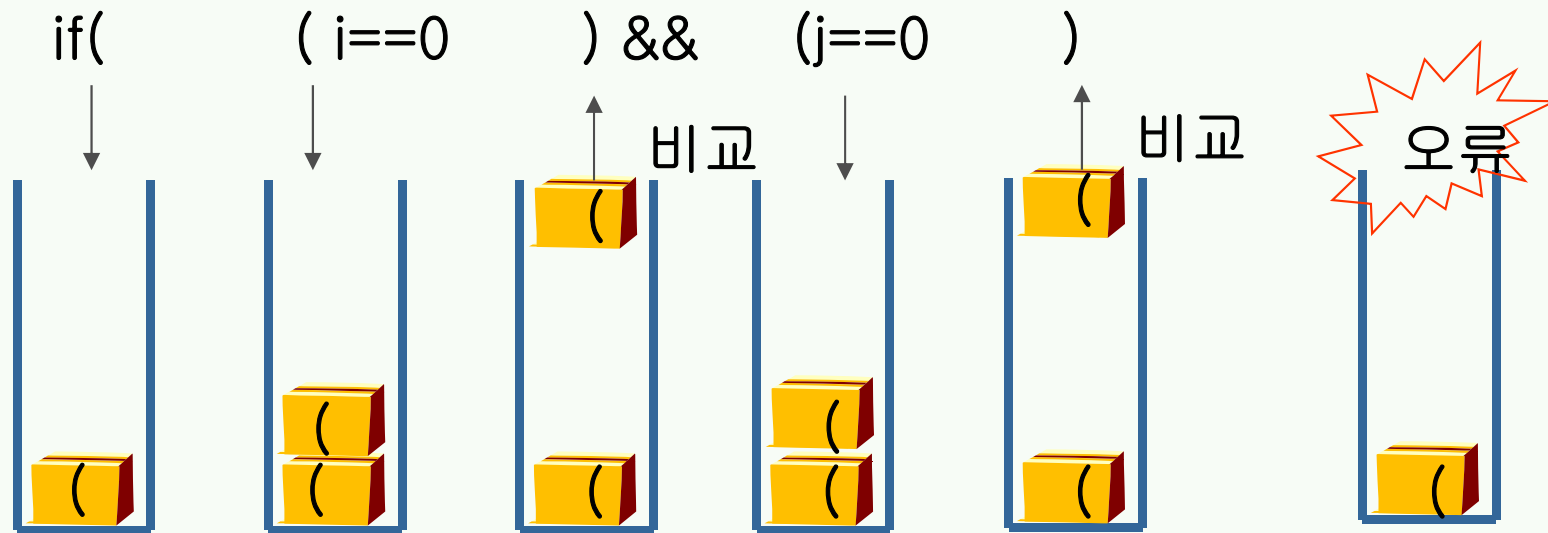
스택의 응용 - 수식에 사용된 괄호 검사

■ 괄호의 검사 조건

- ❖ 조건 1: 왼쪽의 괄호 수와 오른쪽의 괄호 수의 일치
- ❖ 조건 2: 같은 타입의 괄호에서 왼쪽 괄호는 오른쪽 괄호 보다 먼저 나타나야 함
- ❖ 조건 3: 서로 다른 타입의 왼쪽 괄호와 오른쪽 괄호 쌍은 서로 교차하면 안됨

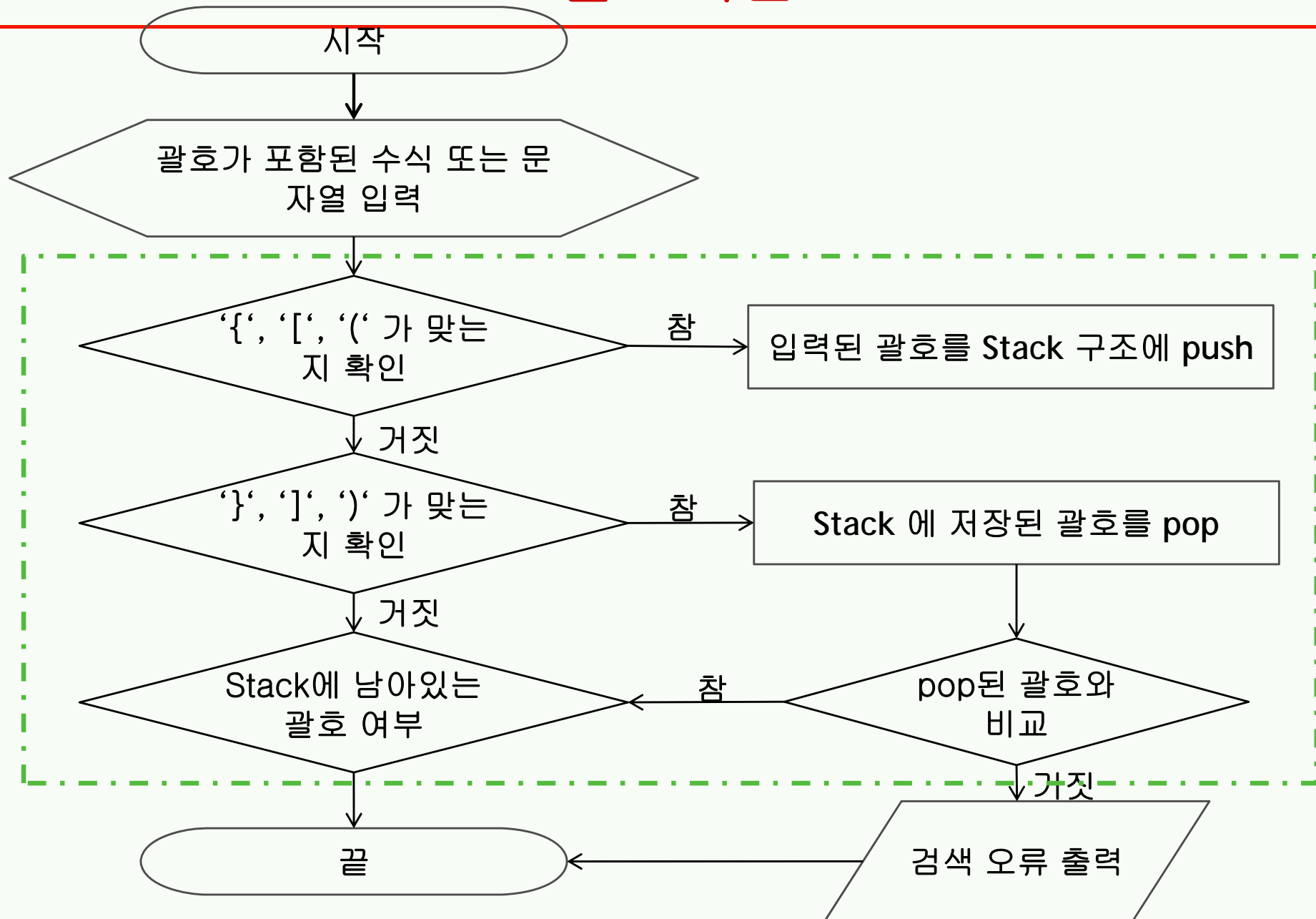
- ❖ 조건 1: $[3 + \{4 * 5 * (6 - 2)\}]$
- ❖ 조건 3: $[4 + \{7 + 2\}]$

스택을 이용한 괄호 검사



그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

알고리즘



파이썬 코드

```
def stack_bracket(text):
```

```
    stack = []
```

```
    for i in text:
```

```
        open_brk = '(['
```

```
        close_brk = ')]'
```

```
        if i in open_brk:
```

```
            stack.append(i)
```

```
        elif i in close_brk:
```

```
            if open_brk.find(stack.pop()) != close_brk.find(i):
```

```
                return False
```

```
    if not stack:
```

```
        return True
```

```
    else:
```

```
        return False
```

```
if __name__ == '__main__':
```

```
    exp = input("input expression >> ")
```

```
    print(stack_bracket(exp))
```

스택을 이용한 수식의 계산

■ 수식의 표기방법:

- 전위(prefix), 중위(infix), 후위(postfix)

중위 표기법	전위 표기법	후위 표기법
$2+3*4$	$+2*34$	$234*+$
$a*b+5$	$+5*ab$	$ab*5+$
$(1+2)+7$	$+7+12$	$12+7+$

■ 컴퓨터에서의 수식 계산순서

- 중위표기식 → 후위표기식 → 계산
- $2+3*4 \rightarrow 234*+ \rightarrow 14$
- 모두 스택을 사용
- 먼저 후위표기식의 계산법을 알아보자

전위표기법

1. 중위표기식에 연산순서로 ()를 추가
2. 제일안쪽의 (괄호 밖으로 연산자를 꺼냄

후위표기법

- 1.은 동일하며, 제일 안쪽의) 뒤로 연산자를 꺼냄

중위표기식->후위표기식

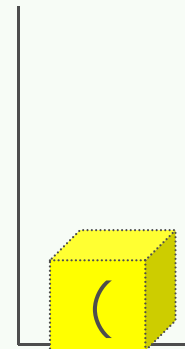
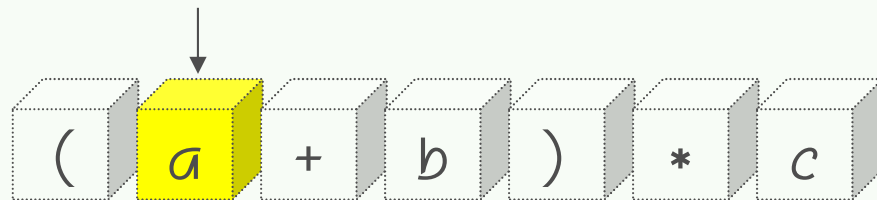
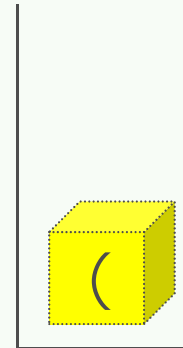
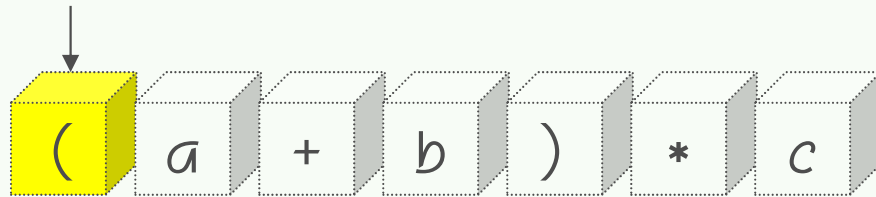
■ 중위표기와 후위표기

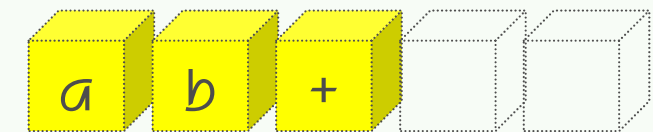
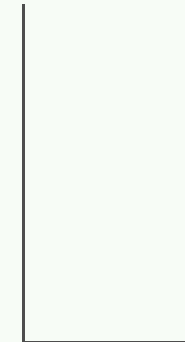
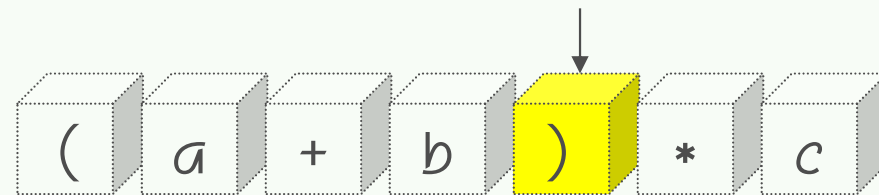
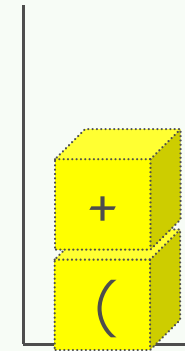
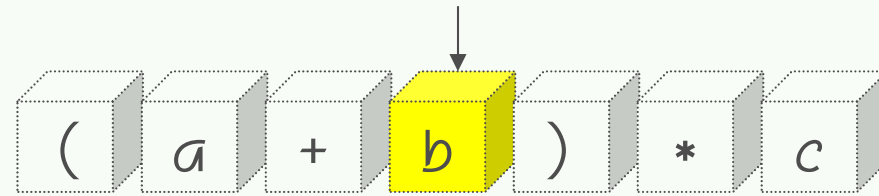
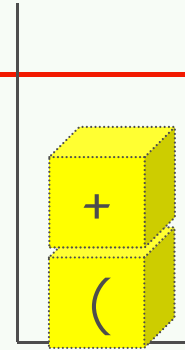
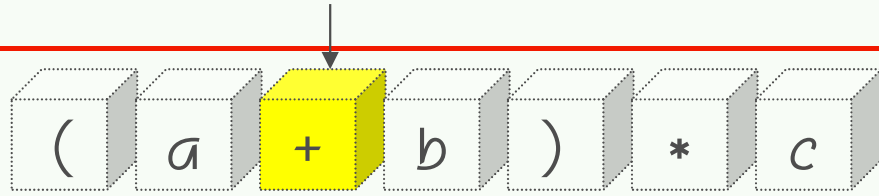
- 중위 표기법과 후위 표기법의 공통점은 피연산자의 순서는 동일
- 연산자들의 순서만 다름(우선순위순서)
->연산자만 스택에 저장했다가 출력하면 된다.
- $2+3*4 \rightarrow 234*+$

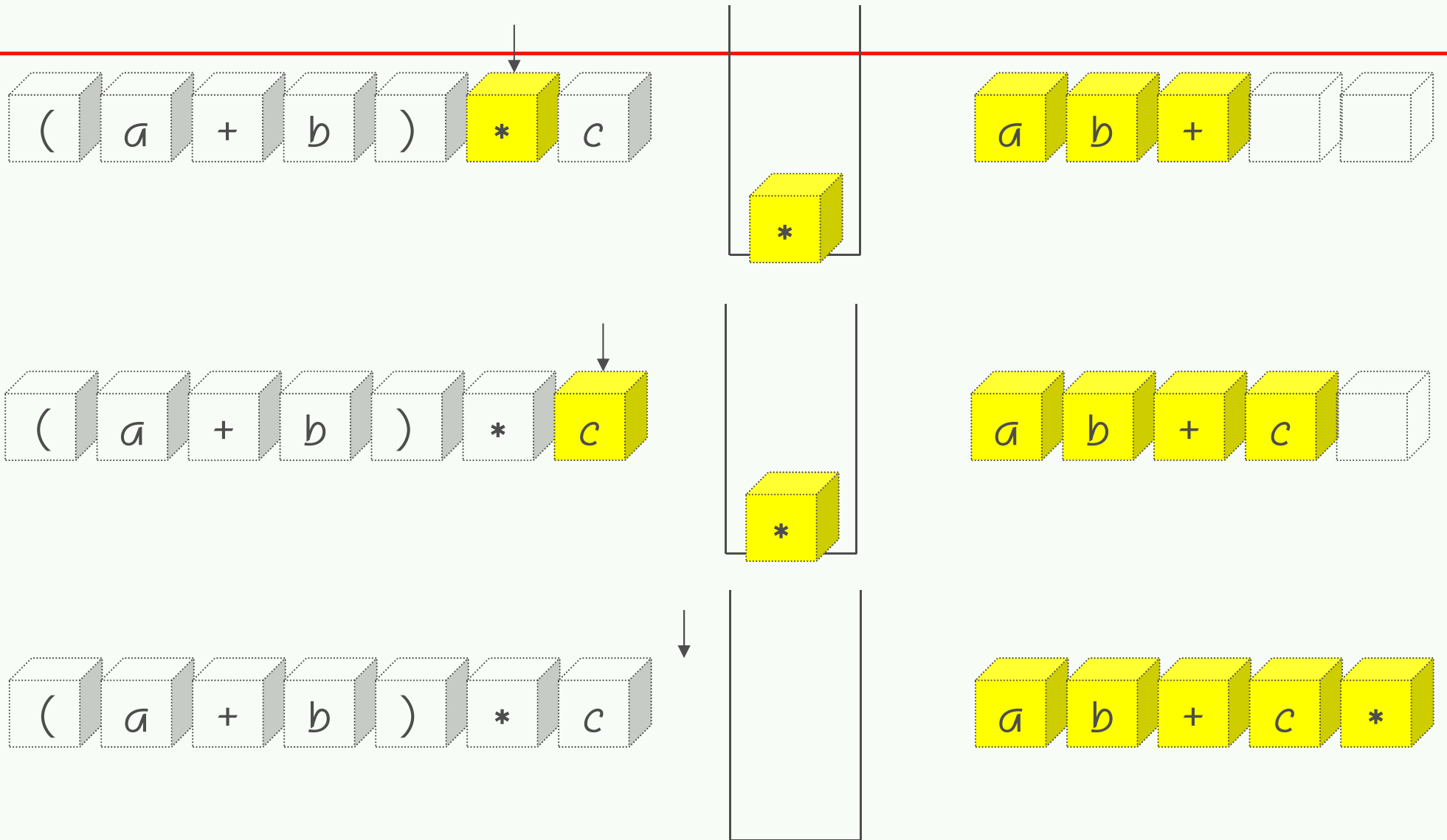
■ 알고리즘

- 피연산자를 만나면 그대로 출력
- 연산자를 만나면 스택에 저장했다가 스택보다 우선 순위가 같거나 낮은 연산자가 나오면 그때 출력
- 괄호에 입력된 연산
 - 왼쪽 괄호는 우선순위가 가장 낮은 연산자로 취급
 - 오른쪽 괄호가 나오면 스택에서 왼쪽 괄호위에 쌓여있는 모든 연산자를 출력

$$(a + b) * C$$



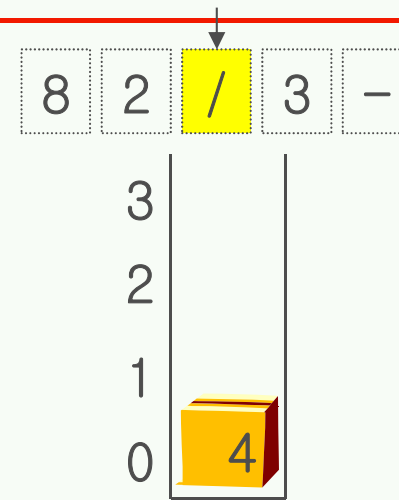
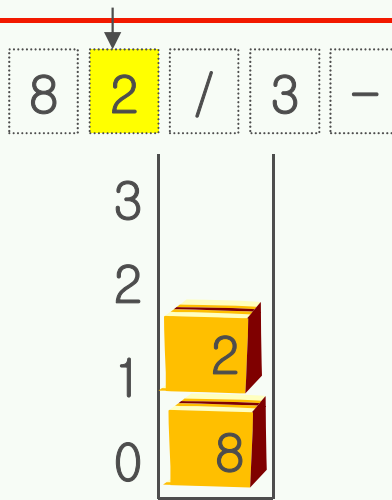
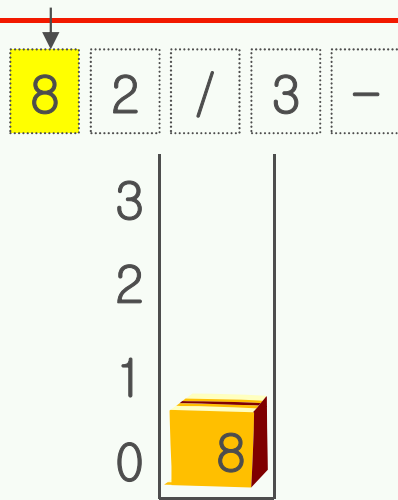




후위 표기식의 계산

- 중위 → 후위로 변환되었다는 가정하에
 - 수식을 왼쪽에서 오른쪽으로 스캔하여 피연산자이면 스택에 저장
 - 연산자이면 필요한 수만큼의 피연산자를 스택에서 꺼내 연산을 실행
 - 연산의 결과를 다시 스택에 저장
 - (예) $82/3-32^*+$

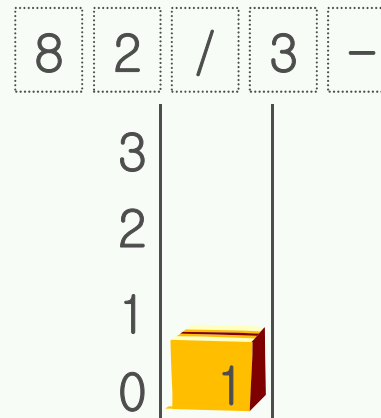
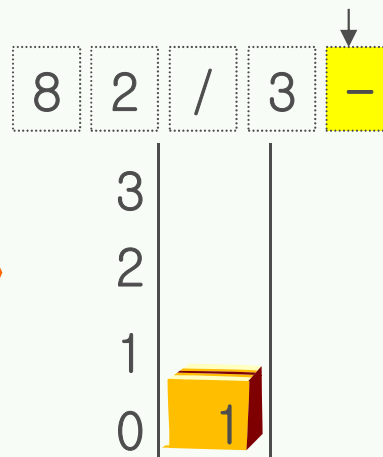
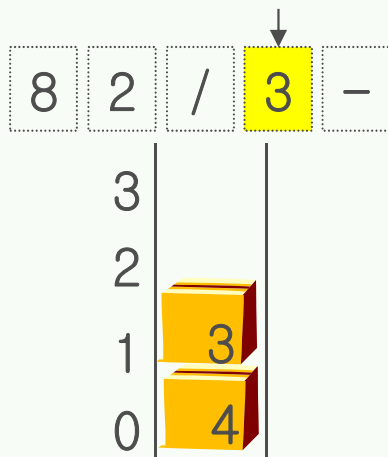
토큰	스택						
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
8	8						
2	8	2					
/	4						
3	4	3					
-	1						
3	1	3					
2	1	3	2				
*	1	6					
+	7						



피연산자-> 삽입

피연산자-> 삽입

연산자-> $8/2=4$ 삽입



피연산자-> 삽입

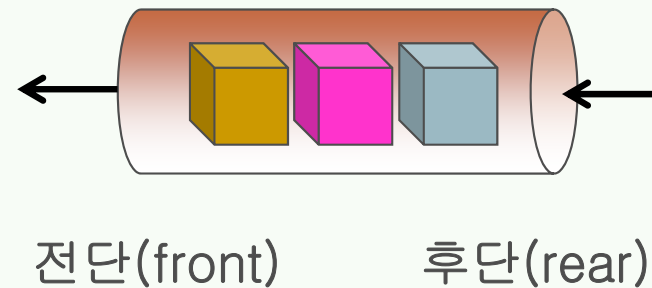
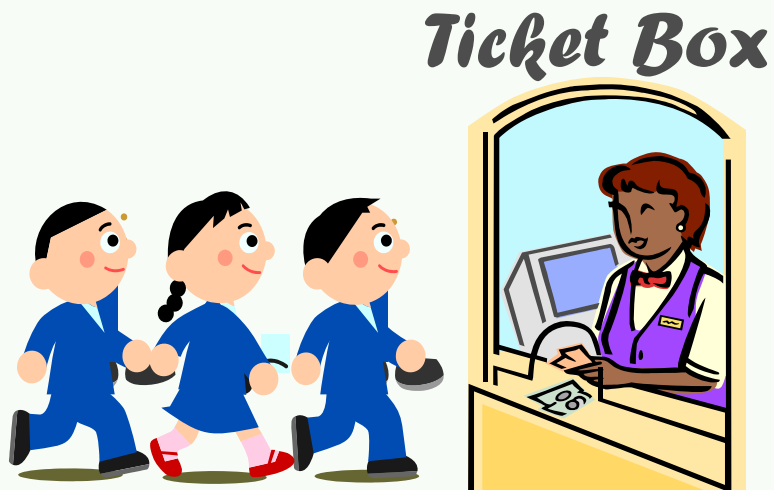
연산자-> $4-1=1$ 삽입

종료->전체 연산 결과=1

그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

큐(Queue)

- 큐: 먼저 들어온 데이터가 먼저 나가는 자료구조
- 선입선출(FIFO: First-In First-Out)
- (예)매표소의 대기열



큐 ADT

- 삽입과 삭제는 FIFO순서를 따름
- 삽입은 큐의 후단에서, 삭제는 전단에서 이루어짐

·객체: n개의 element형으로 구성된 요소들의 순서있는 모임

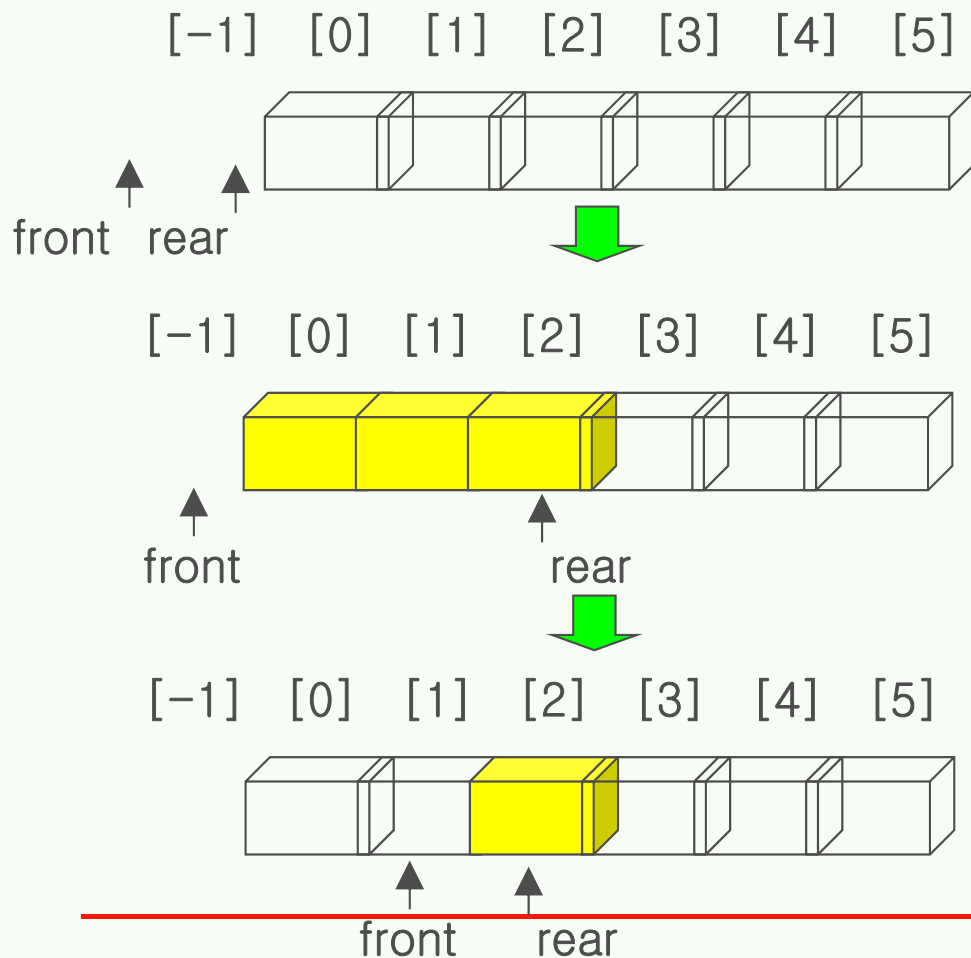
·연산:

- `create()` ::= 큐를 생성한다.
- `init(q)` ::= 큐를 초기화한다.
- `is_empty(q)` ::= 큐가 비어있는지를 검사한다.
- `is_full(q)` ::= 큐가 가득 찼는가를 검사한다.
- `enqueue(q, e)` ::= 큐의 뒤에 요소를 추가한다.
- `dequeue(q)` ::= 큐의 앞에 있는 요소를 반환한 다음 삭제한다.
- `peek(q)` ::= 큐에서 삭제하지 않고 앞에 있는 요소를 반환한다.

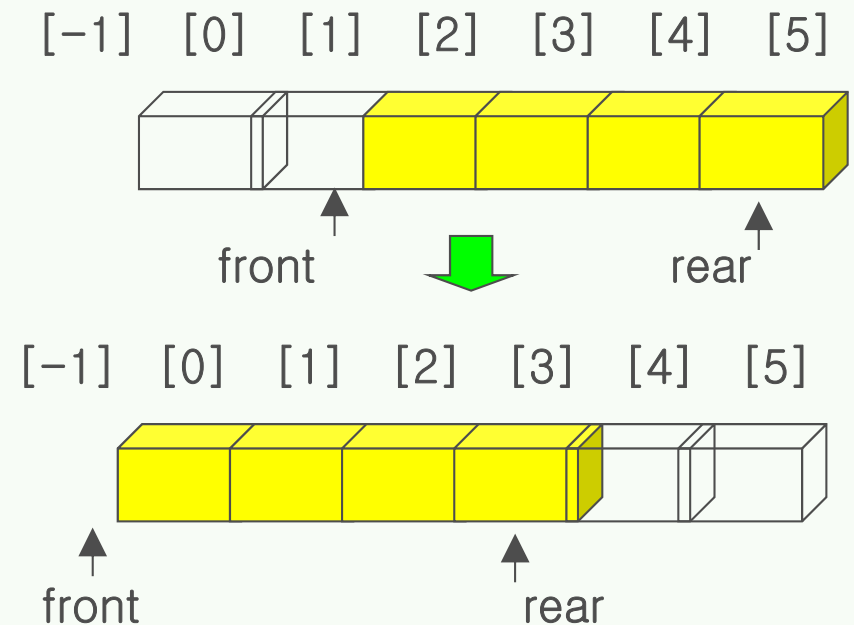
배열을 이용한 선형큐

■ 선형큐: 배열을 선형으로 사용하여 큐를 구현

- 삽입을 계속하기 위해서는 요소들을 이동시켜야 함
- 문제점이 많아 사용되지 않음



왼쪽으로
데이터
이동



파이썬 코드

```
def enqueue(que,item):
    que.append(item)

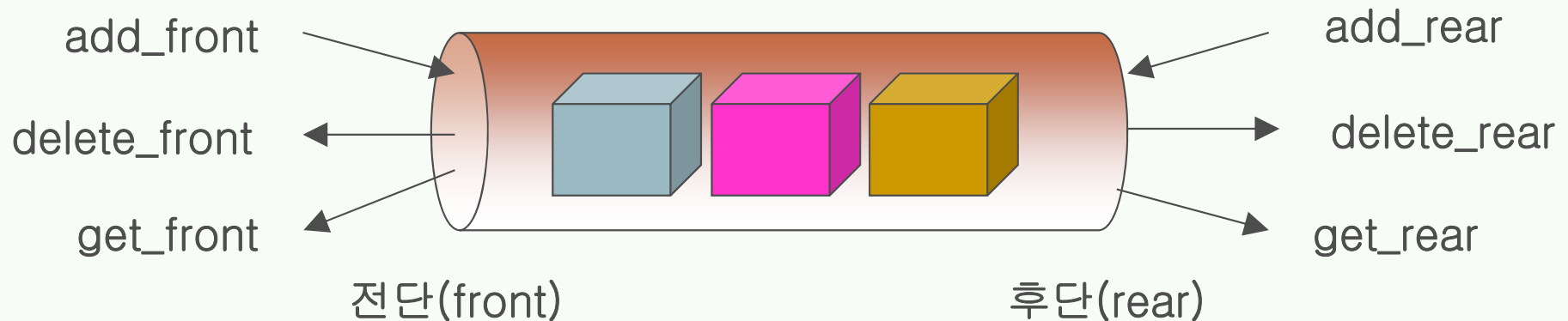
def dequeue(que):
    if len(que) !=0:
        que.pop(0)
    else:
        print("Empty Queue")

def queue_print(que):
    print('front --> ', end='')
    for i in range(len(que)):
        print(f'{que[i]}', end='\t')
    print('<-- rear')
```

```
if __name__ == "__main__":
    q = []
    enqueue(q,'apple')
    enqueue(q,'banana')
    enqueue(q,'grape')
    queue_print(q)
    dequeue(q)
    dequeue(q)
    queue_print(q)
```

덱(deque)

- 덱(deque)은 double-ended queue의 줄임말로써 큐의 전단(front)와 후단(rear)에서 모두 삽입과 삭제가 가능한 큐
- 스택과 큐를 동시에 구현하는데 사용
- 사용예: 문서 편집기의 undo, 웹 브라우저의 방문 기록 등



그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

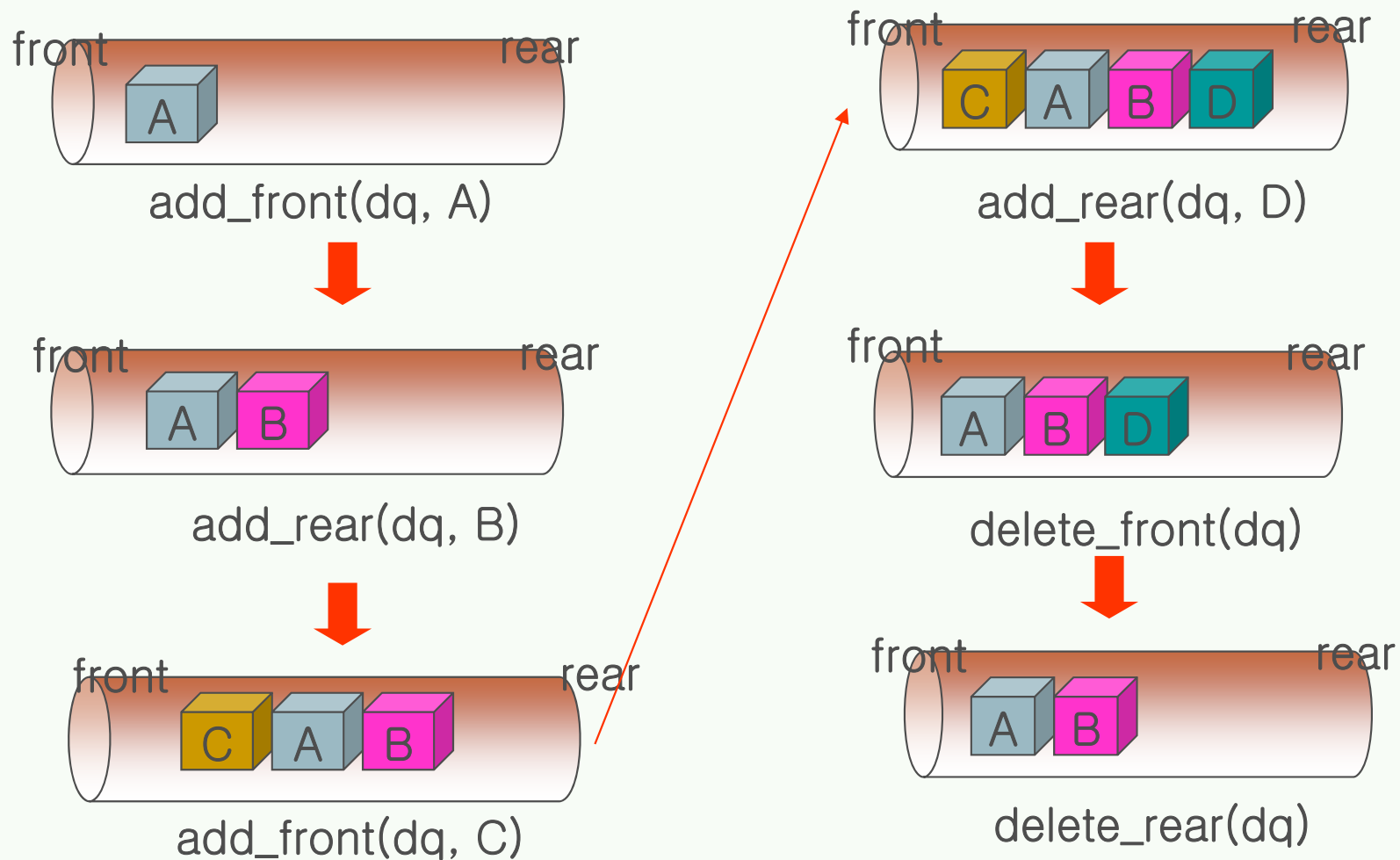
덱의 ADT

·객체: n개의 element형으로 구성된 요소들의 순서있는 모임

·연산:

- `create()` ::= 덱을 생성한다.
- `init(dq)` ::= 덱을 초기화한다.
- `is_empty(dq)` ::= 덱이 공백상태인지를 검사한다.
- `is_full(dq)` ::= 덱이 포화상태인지를 검사한다.
- `add_front(dq, e)` ::= 덱의 앞에 요소를 추가한다.
- `add_rear(dq, e)` ::= 덱의 뒤에 요소를 추가한다.
- `delete_front(dq)` ::= 덱의 앞에 있는 요소를 반환한 다음 삭제한다
- `delete_rear(dq)` ::= 덱의 뒤에 있는 요소를 반환한 다음 삭제한다.
- `get_front(q)` ::= 덱의 앞에서 삭제하지 않고 앞에 있는 요소를 반환한다.
- `get_rear(q)` ::= 덱의 뒤에서 삭제하지 않고 뒤에 있는 요소를 반환한다.

덱의 연산



그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

파이썬 코드

- collections 패키지의 deque 모듈을 통해 구현

```
from collections import deque

dq = deque()
print()
item = ['berry', 'banana']
dq.append('apple')
dq.appendleft('pear')
dq.appendleft(item)
print(dq)
dq.pop()
dq.popleft()
print(dq)
item = ['cherry', 'mango']
dq.extend(item)
print(dq)
```

큐의 응용: 버퍼

- 큐는 서로 다른 속도로 실행되는 두 프로세스 간의 상호 작용을 조화시키는 버퍼 역할을 담당
 - CPU와 프린터 사이의 프린팅 버퍼, 또는 CPU와 키보드 사이의 키보드 버퍼
- 대개 데이터를 생산하는 생산자 프로세스가 있고 데이터를 소비하는 소비자 프로세스가 있으며 이 사이에 큐로 구성되는 버퍼가 존재

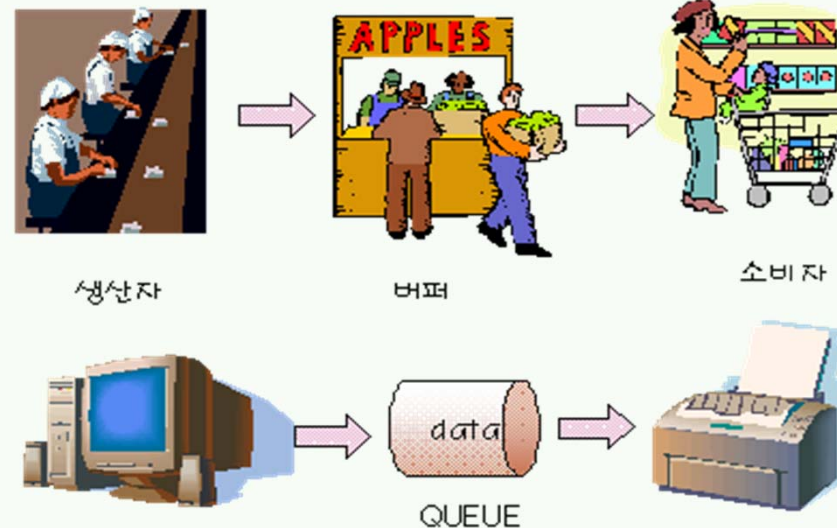
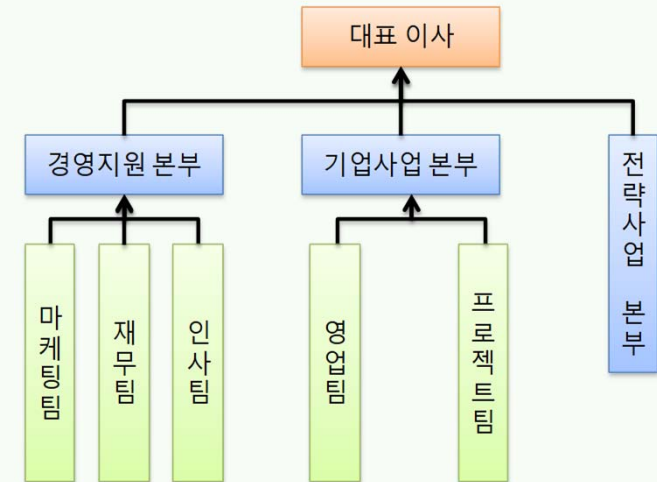


그림 6.17 생산자와 버퍼, 소비자의 개념 그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

트리(TREE)

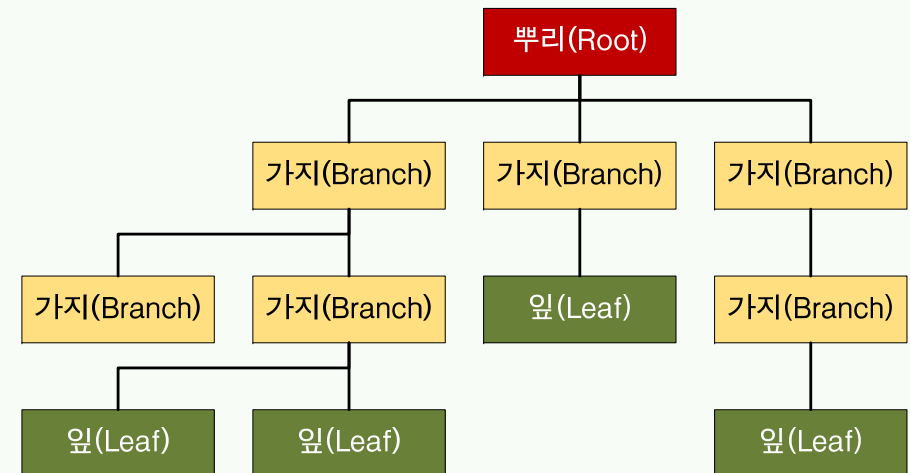
■ 트리: 노드(Node) + 간선(Edge)

- 계층적인 구조를 나타내는 자료구조
- 비선형 구조
- 트리는 부모-자식 관계의 노드들로 이루어짐



■ 응용분야:

- 계층적인 조직 표현
- 파일 시스템
- 검색 엔진
- 인공지능에서의 결정트리



그림출처: 열혈강의 - C로 만드는 자료구조 및 알고리즘

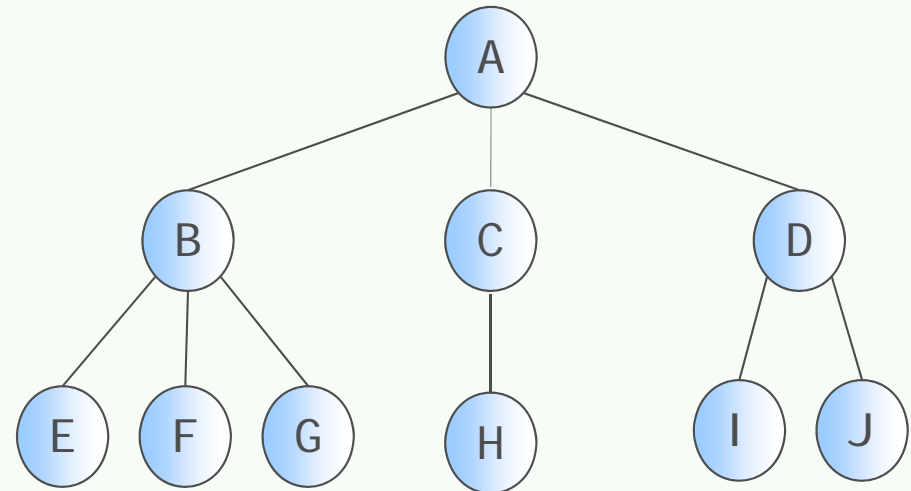
트리의 용어(1)

■ 트리에서의 위치에 따라

- 루트(root) 노드: 부모가 없는 노드, 트리의 첫 노드
- 단말(leaf 또는 terminal) 노드: 자식 노드가 없는 노드
- 내부(internal) 노드: 자식 노드가 있는 노드

루트: A

단말: E, F, G, H, I, J



****노드(node):** 트리의 구성요소

**** 간선(edge):** 노드와 노드의 연결선

트리의 용어(2)

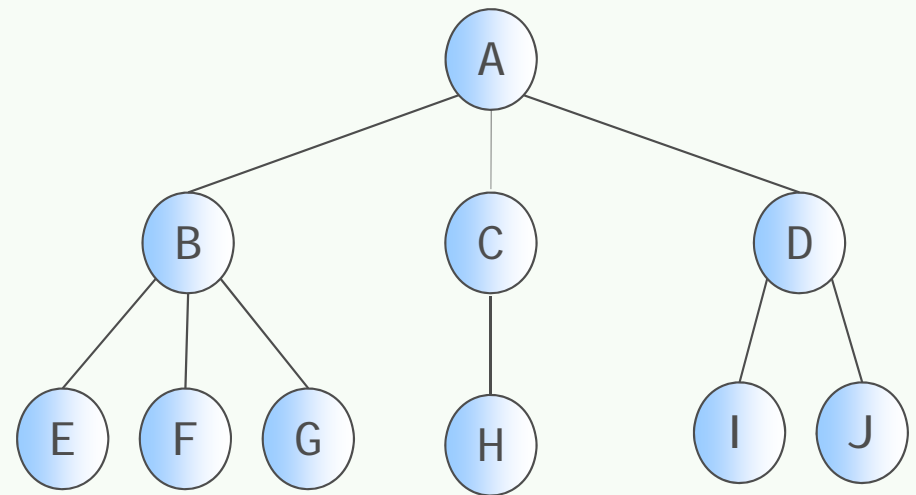
■ 노드 사이의 관계에 따라

- 부모(parent) 노드
- 자식(child) 노드
- 선조(ancestor) 노드 : 루트 노드부터 부모 노드까지의 경로 상에 있는 모든 노드
- 후손, 자손(descendant) 노드: 특정 노드의 아래에 있는 모든 노드
- 형제(sibling) 노드: 같은 부모 노드의 자식 노드

B, C, D의 부모 노드: A

B의 형제 노드: C, D

H의 선조 노드: C, A



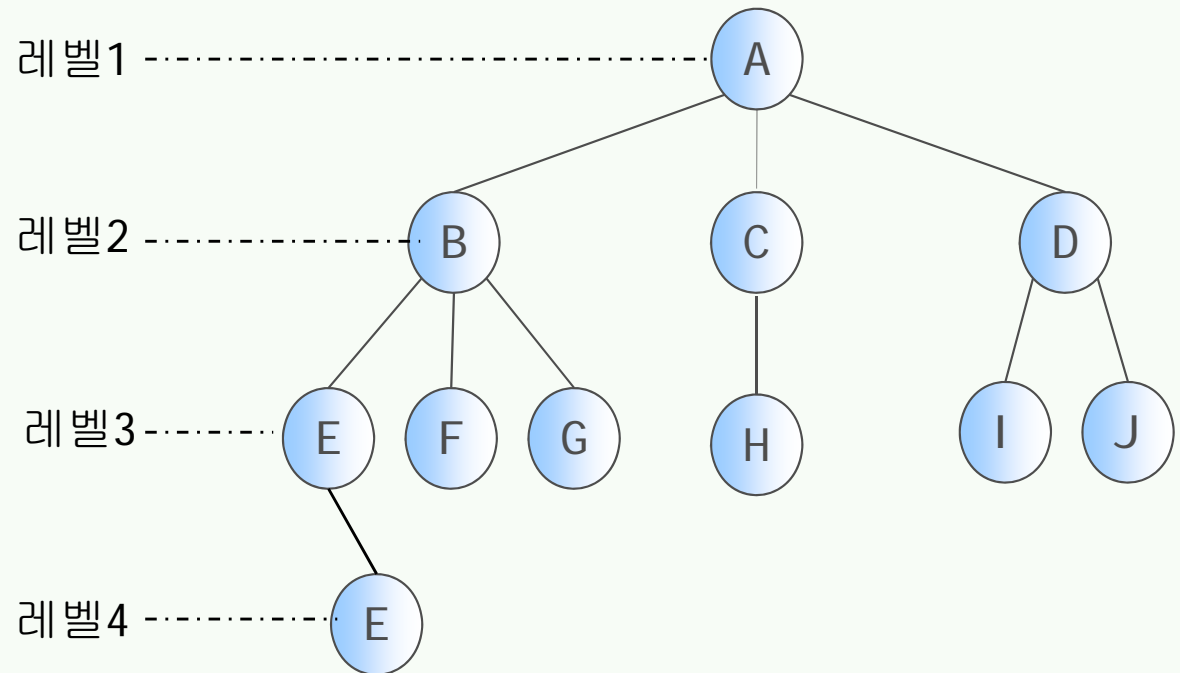
트리의 용어(3)

■ 노드의 속성

- 레벨(level): 루트 노드로부터의 거리
- 높이(height): 루트 노드부터 가장 먼 거리에 있는 자식 노드의 높이
- 차수(degree): 한 노드가 가지는 자식 노드의 개수

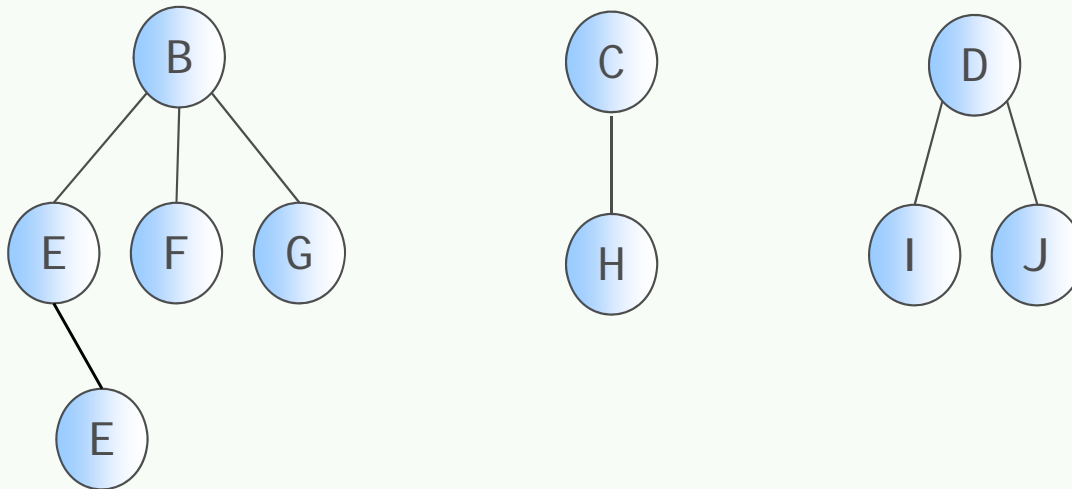
B의 높이: 3

D의 차수: 2



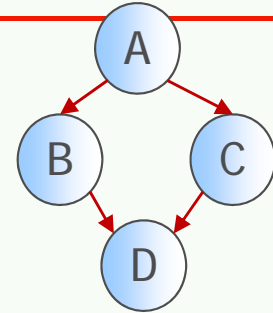
트리의 용어(4)

- 포레스트(forest): 트리의 집합, 루트 노드가 여러 개 일 수 있음



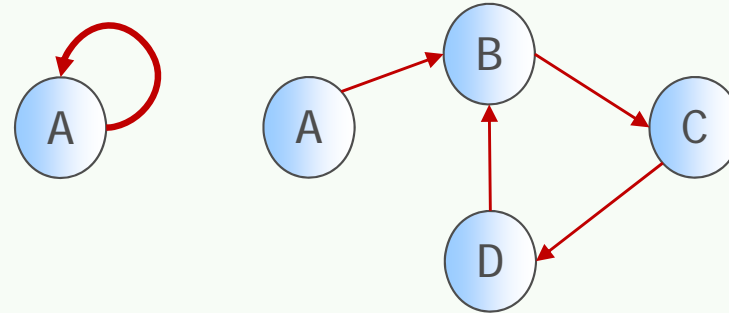
트리의 성질

- 한 노드에서 다른 노드로 가는 경로는 유일
- N개의 노드를 갖는 나무는 N-1개의 간선(edge)을 가짐

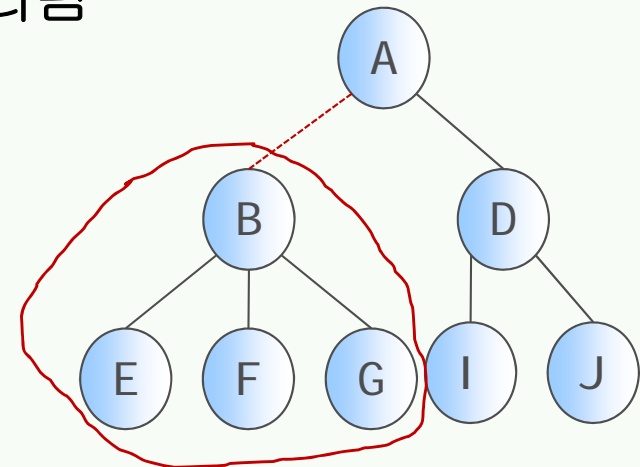


- 뿌리 노드(root node)를 제외하고 모든 노드가 자신의 선주를 향해 하나의 링크를 가짐

- 순환(cycle)이 존재하지 않음
- 모든 노드는 연결되어 있음

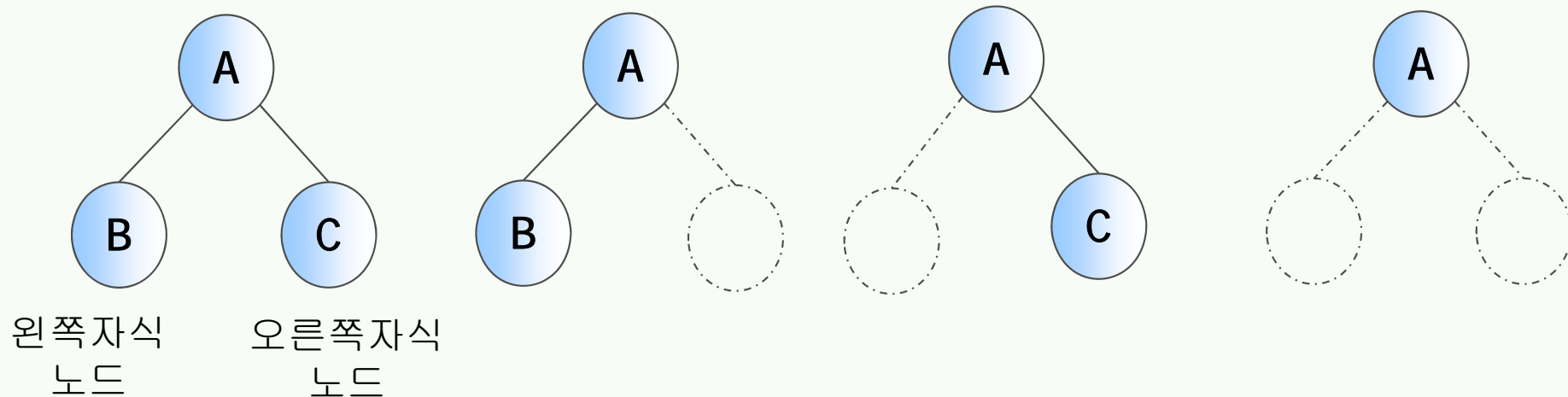


- 간선(edge)을 하나 자르면 두 개의 트리로 분리됨



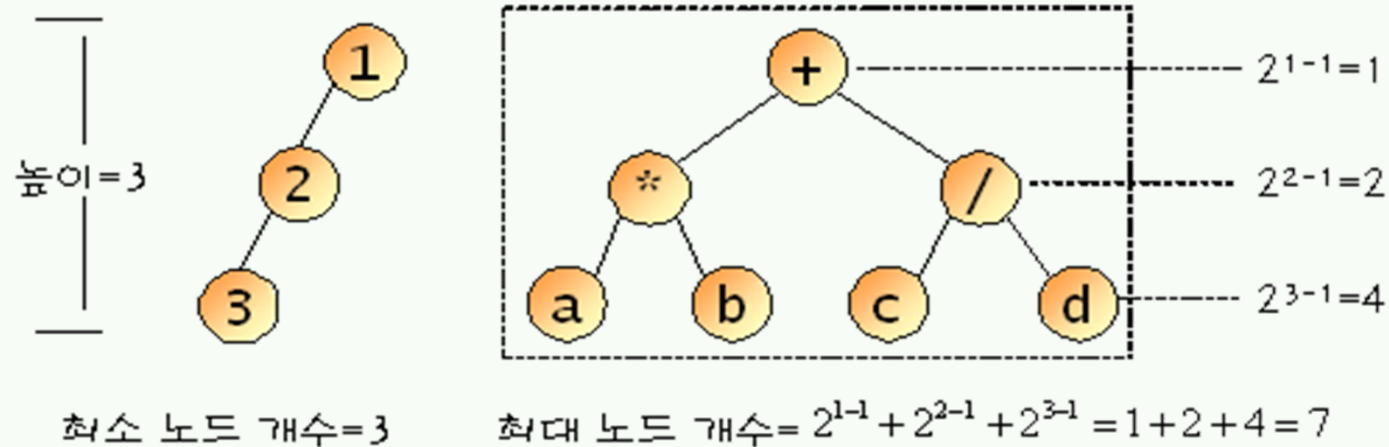
이진트리(binary tree)

- 이진 트리(binary tree) : 모든 노드가 최대 2개의 서브 트리를 가지고 있는 트리
 - 서브트리는 공집합일수 있다.
- 모든 노드의 차수가 2 이하 -> 구현하기가 편리함
- 이진 트리에는 서브 트리간의 순서가 존재



이진트리의 성질

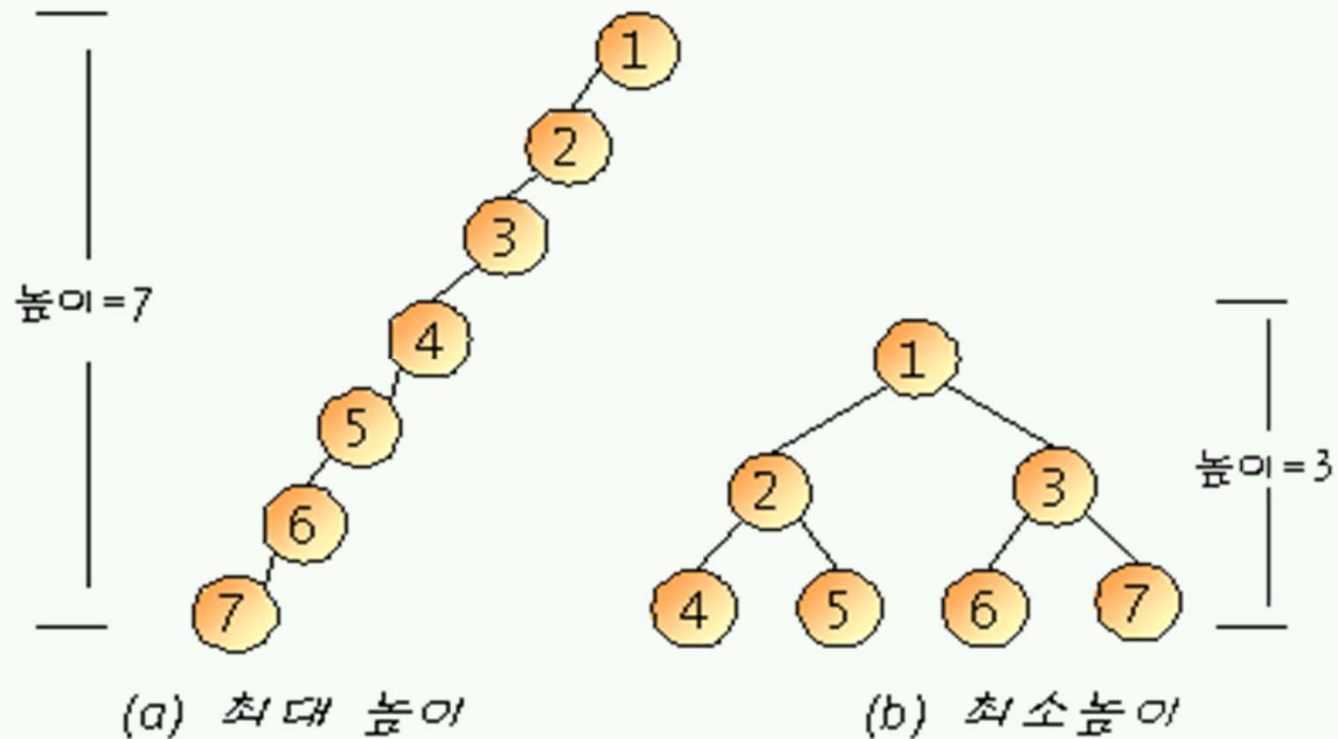
- 노드의 개수가 n 개이면 간선의 개수는 $n-1$
- 높이가 h 인 이진트리의 경우, 최소 h 개의 노드를 가지며 최대 2^h-1 개의 노드를 가짐



- n 개의 노드를 가지는 이진 트리의 높이
 - 최대 n 개의 높이이거나,
 - 최소 $\lceil \log_2(n+1) \rceil$ 개의 높이를 가짐, $\lceil \cdot \rceil$: 소수점 올림 연산자 (*ceil* 함수)

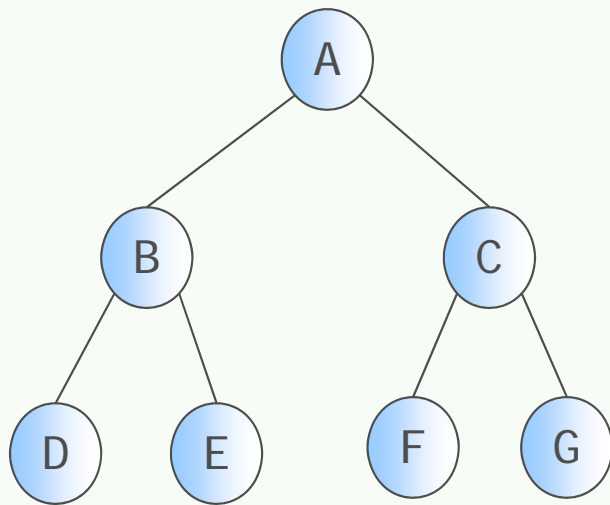
이진트리의 성질

- n 개의 노드를 가지는 이진트리의 높이는 최대 n 이거나 최소 $\lceil \log_2(n+1) \rceil$

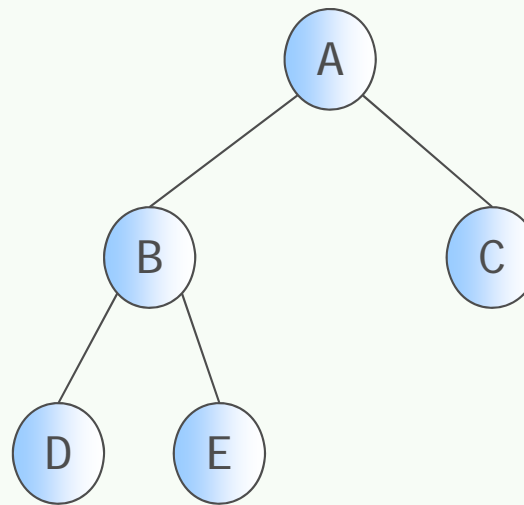


이진트리의 분류

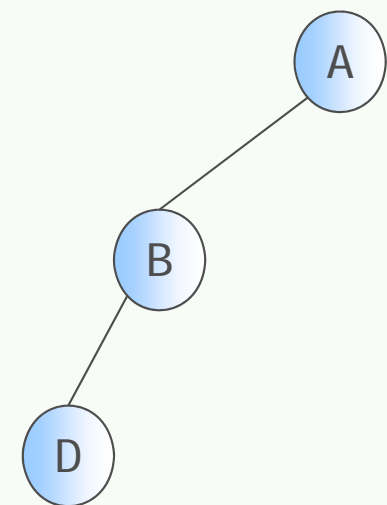
- 포화 이진 트리(full binary tree): 트리의 각 레벨에 노드가 꽉 차있는 이진트리
- 완전 이진 트리(complete binary tree):
 - 높이가 h 일 때 레벨 1부터 $h-1$ 까지는 노드가 모두 채워져 있고
 - 마지막 레벨 h 에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져 있는 이진트리
- 편향 이진 트리(skewed binary tree)



<포화이진트리>



<완전이진트리>



<편향이진트리>

이진 트리의 ADT(추상 자료형)

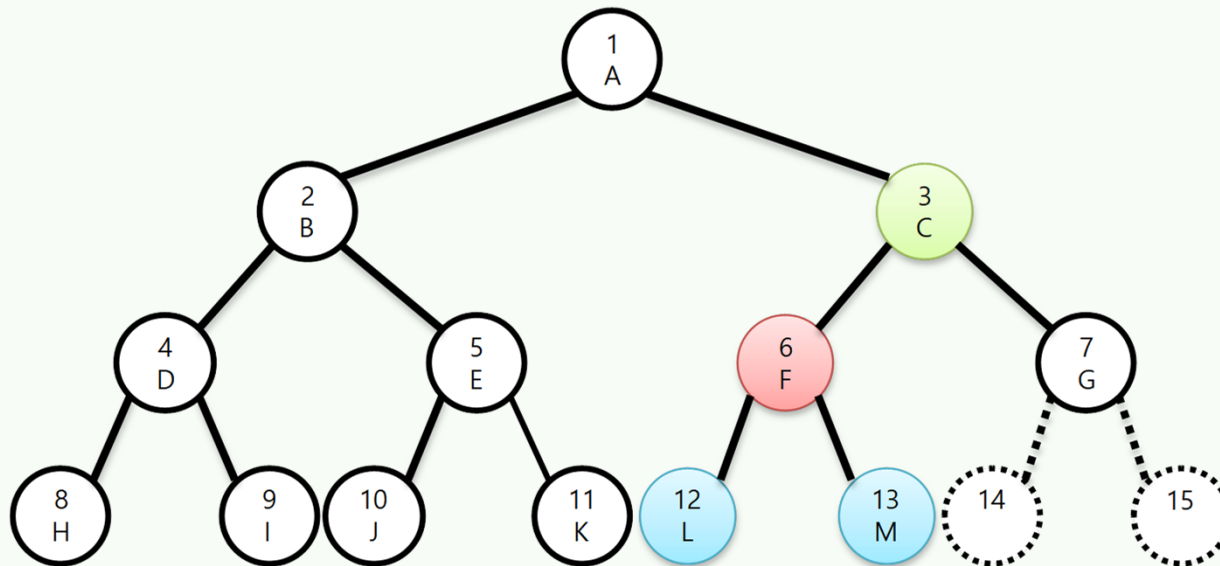
- ❖ 이진트리 생성
- ❖ 이진트리 삭제
- ❖ 루트 노드 반환
- ❖ 왼쪽 자식 노드 추가
- ❖ 오른쪽 자식 노드 추가
- ❖ 왼쪽 자식 노드 반환
- ❖ 오른쪽 자식 노드 반환

```
BinTree* makeBinTree(BinTreeNode rootNode);  
BinTreeNode* getRootNode(BinTree* pBinTree);  
BinTreeNode* insertLeftChild(BinTreeNode* pParentNode, binTreeNode item);  
BinTreeNode* insertRightChild(BinTreeNode* pParentNode, binTreeNode item);  
BinTreeNode* getLeftChild(BinTreeNode* pNode);  
BinTreeNode* getRightChild(BinTreeNode* pNode);  
void deleteBinTree(BinTree* pBinTree);
```

배열을 이용한 이진 트리 생성

■ 노드번호 / 저장 자료 : 노드 j

- 부모 노드 인덱스 $= \lfloor j/2 \rfloor$, 단 $j > 1$
- 왼쪽 자식 노드 인덱스 $= 2 * j$
- 오른쪽 자식 노드 인덱스 $= (2*j) + 1$



0	
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	K
12	L
13	M
14	
15	

그림참조: C로 쉽게 풀어쓴 자료구조, 생능출판사

파이썬 코드

```
bt = [None]*1
```

```
bt.append('A')  
bt.append('B')  
bt.append('C')  
bt.append('D')  
bt.append('E')
```

```
bt = [None]*1
```

```
bt.append([90,'watermelon'])  
bt.append([80,'pear'])  
bt.append([70,'melon'])  
bt.append([50,'lime'])  
bt.append([60,'mango'])
```

링크를 이용한 파이썬 코드

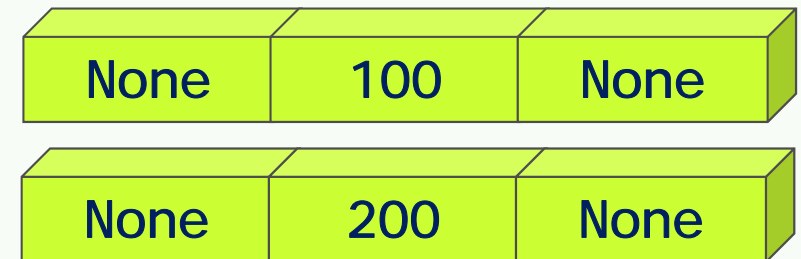
```
class Node:
    def __init__(self, item, left=None, right=None):
        self.item = item
        self.left = left
        self.right = right
```



```
class BinaryTree:
    def __init__(self):
        self.root = None
```



```
if __name__ == '__main__':
    t = BinaryTree()
    value = [100, 200, 300, 400, 500, 600, 700, 800]
    n = []
    for i in value:
        n.append(Node(i))
```



링크를 이용한 파이썬 코드

```
for i in range(len(value)//2):
```

```
    if i == 0:
```

```
        n[i].left = n[i+1]
```

```
        n[i].right = n[i+2]
```

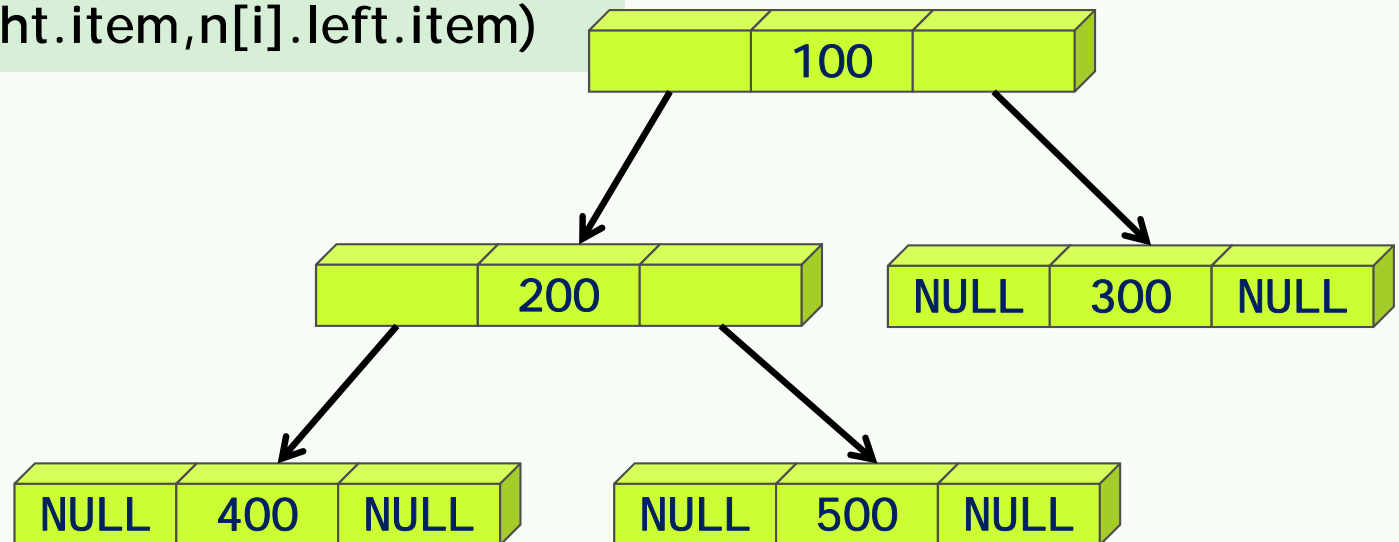
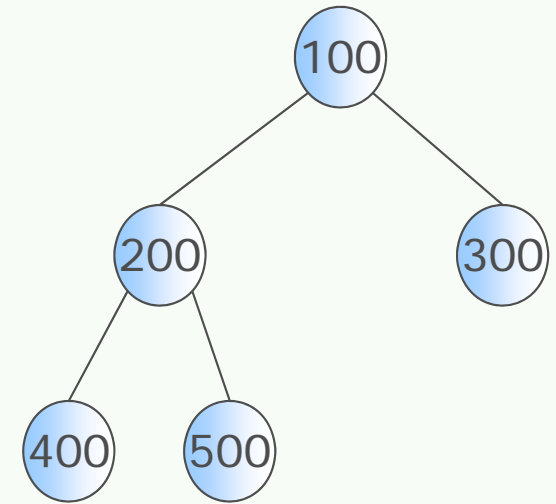
```
    else:
```

```
        n[i].left = n[2*i+1]
```

```
        if 2*i + 2 < len(value)-1:
```

```
            n[i].right = n[2*i+2]
```

```
    #print(n[i].item,n[i].right.item,n[i].left.item)
```



파이썬 데이터처리 및 분석

Numpy 모듈

2020.07. 15. 수요일

최희련

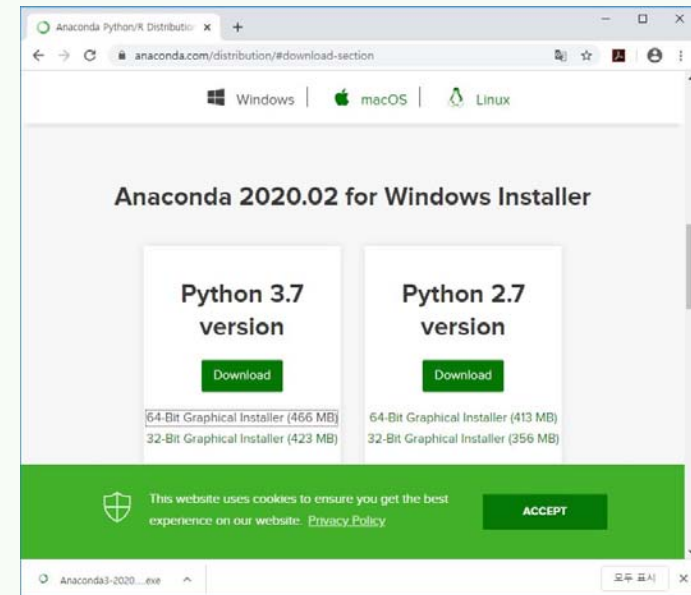
En-CORE

Data Science Edu.

Jupyter notebook 설치

■ Anaconda 설치

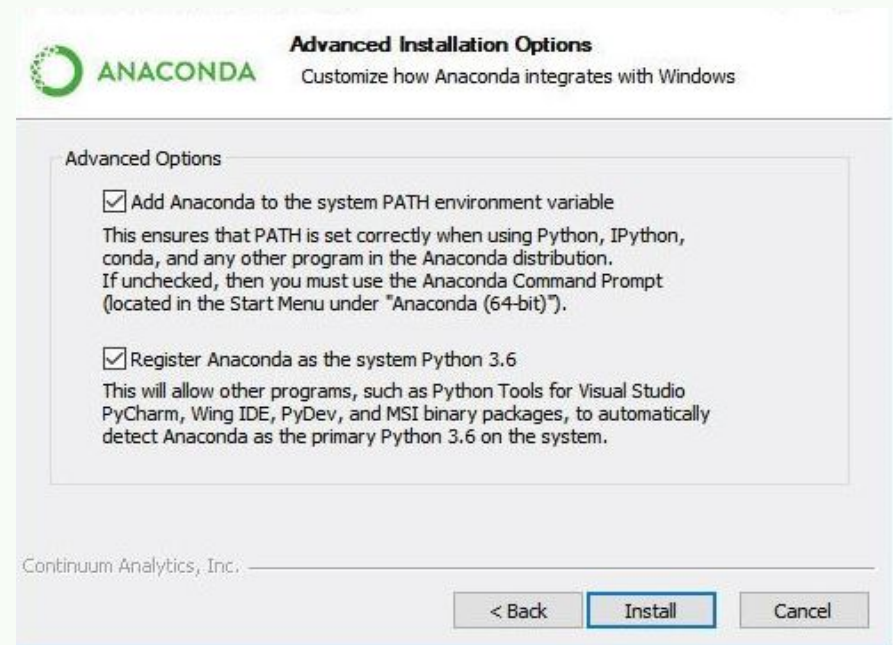
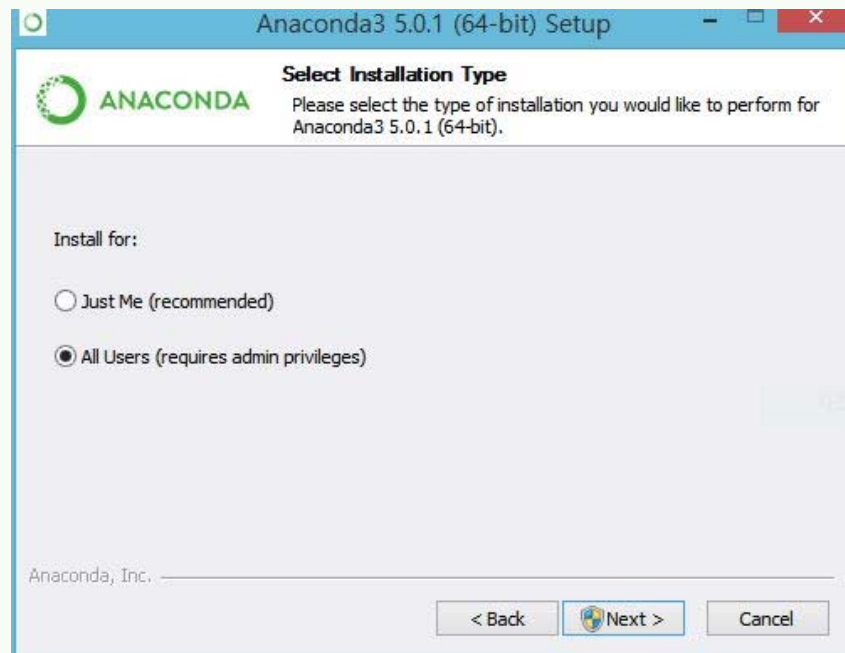
- Anaconda 는 수학, 과학분야의 패키지로 튜닝한 파이썬
- Jupyter notebook, python 이 자동으로 설치
- www.anaconda.com/download
 - Window/macOS/Linux 중, 본인의 컴퓨터 OS에 맞는 설치파일 선택
 - OS 선택 후, 32 또는 64bit 선택 (tensorflow 등의 기계 또는 딥러닝 라이브러리 사용 목적이라면, 64bit 선택)
- 주의, tensorflow 등의 AI 모듈 사용 시
 - Anaconda3 설치 시 파이썬 3.7 설정



Jupyter notebook 설치

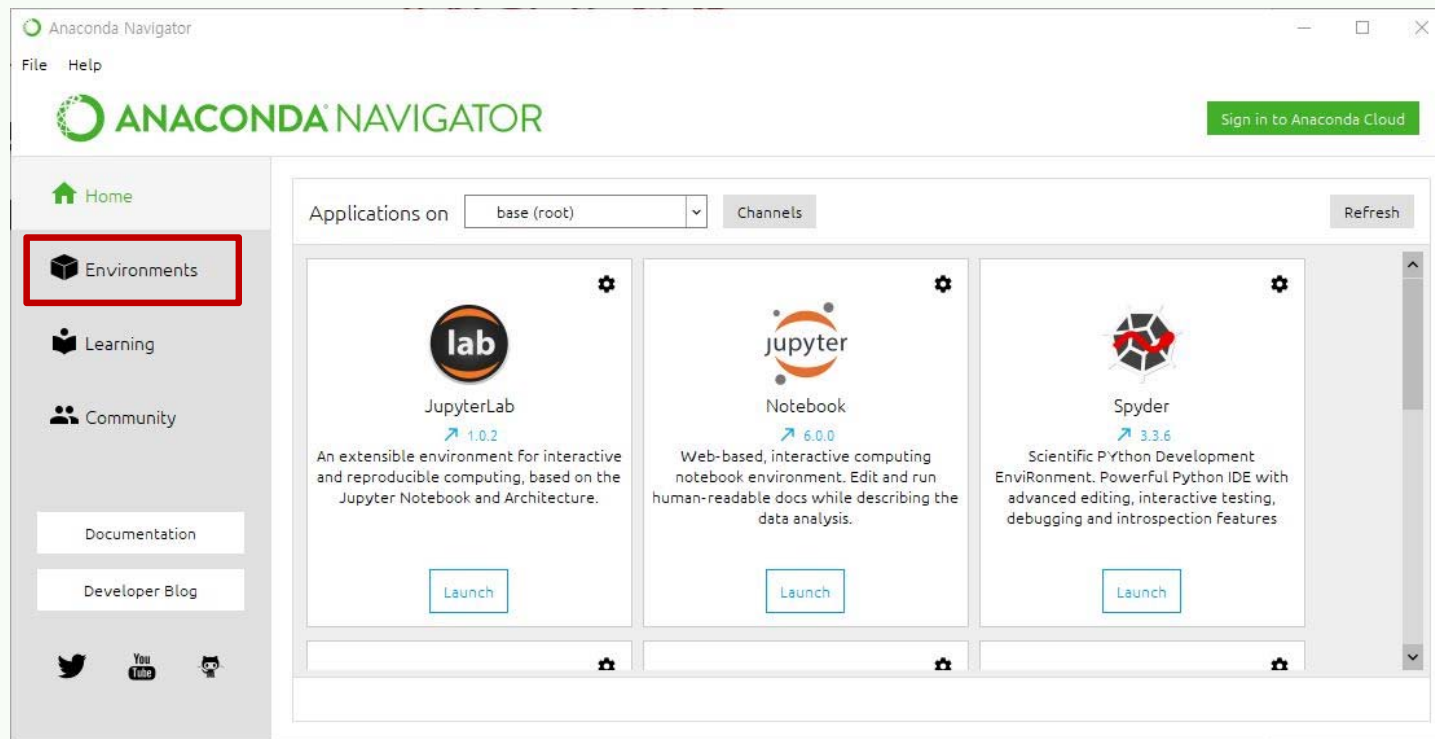
■ Anaconda 설치 단계

- Anaconda License 동의화면에서 동의 선택
- Select Installation Type - 기본적으로 All User 선택
- Choose Install Location - 설치경로 선정
- Advanced Installation Options - 시작 경로 설정



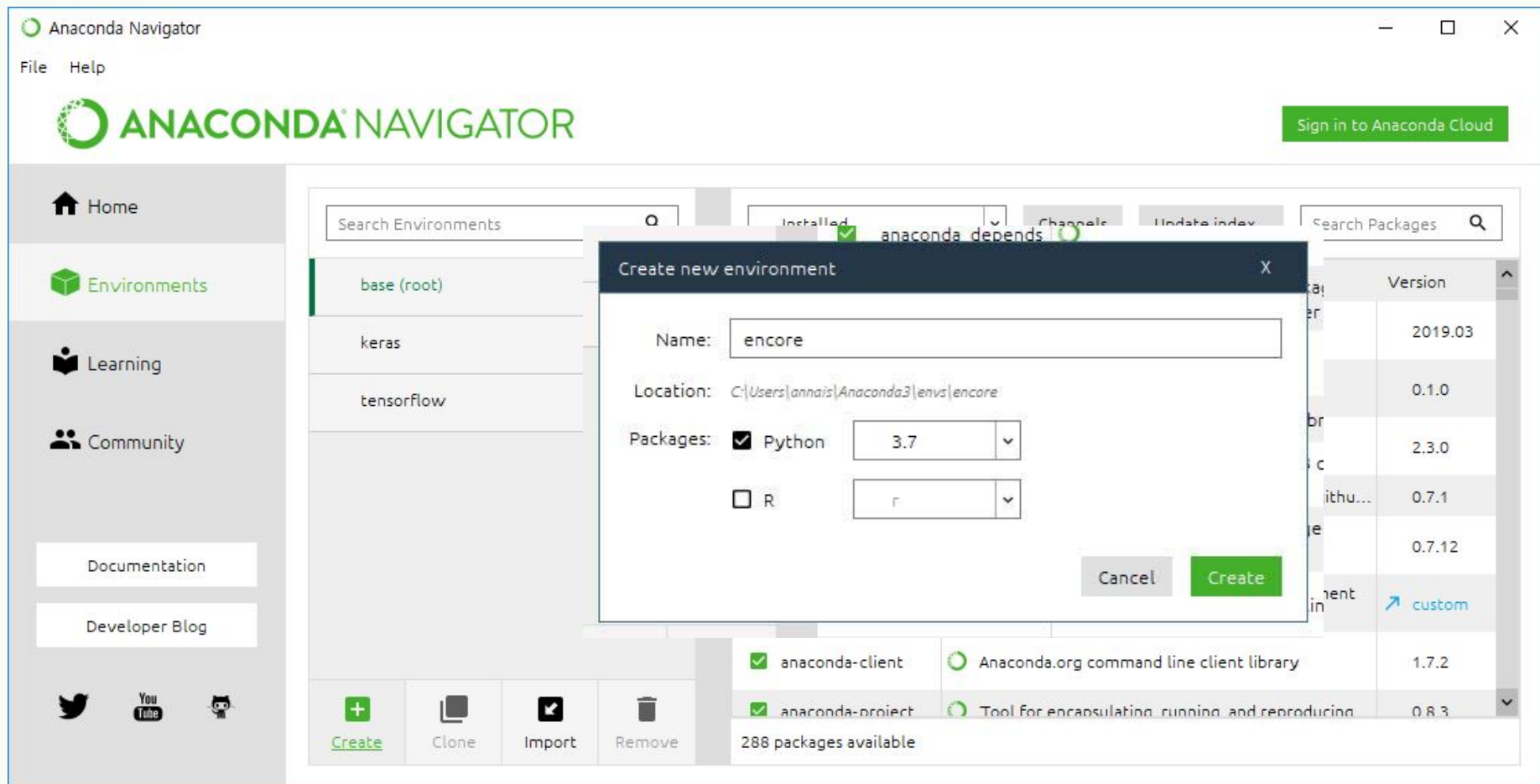
가상환경 설정(1)

- 가상 환경 설정은 python 버전 관리 용이, 패키지의 충돌 방지 등의 장점
- 아나콘다 설치 후, [Anaconda Navigator] 실행
 - 왼쪽의 [Environments] 실행



가상환경 설정(2)

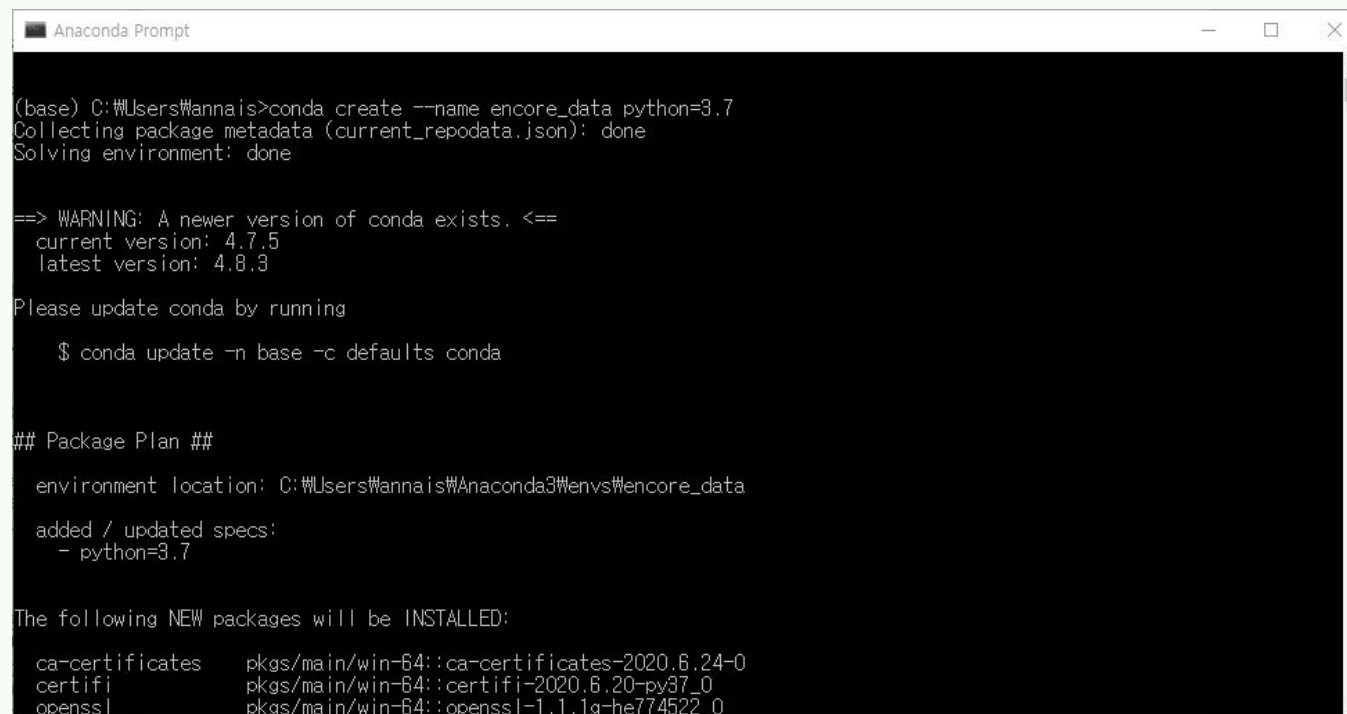
- 화면 하단의 [Create]를 클릭
- 가상환경 이름과 파이썬 버전을 선택 후, 대화창의 Create 버튼 클릭



가상환경 설정(3)

■ Conda prompt 사용 시

- `conda create --name [가상환경이름] python = [연결할 파이썬 버전]`
- `conda create --name encore python=3.7`
- 설정된 가상환경 리스트 확인 → `conda env list` 입력 후 엔터



```
Anaconda Prompt

(base) C:\Users\Wannais>conda create --name encore_data python=3.7
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.7.5
  latest version: 4.8.3

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

  environment location: C:\Users\Wannais\Anaconda3\envs\encore_data

  added / updated specs:
    - python=3.7

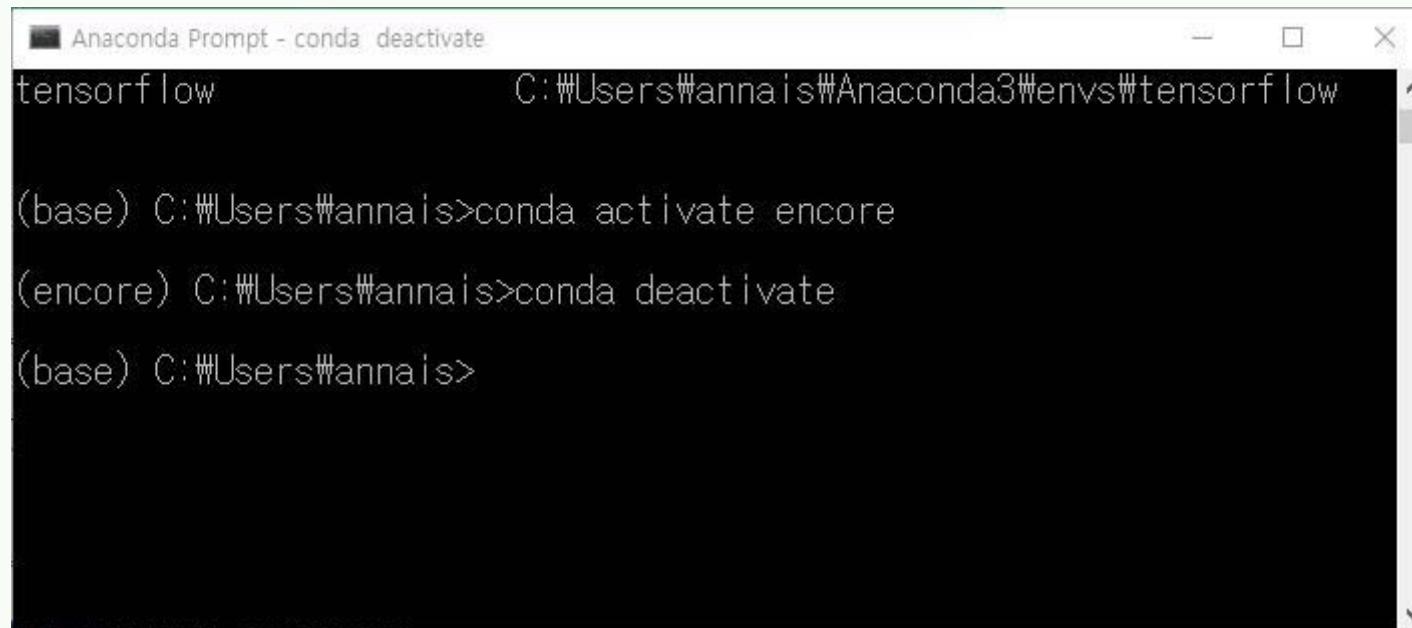
The following NEW packages will be INSTALLED:

  ca-certificates pkgs/main/win-64::ca-certificates-2020.6.24-0
  certifi         pkgs/main/win-64::certifi-2020.6.20-py37_0
  openssl        pkgs/main/win-64::openssl-1.1.1g-he774522_0
```

가상환경 설정(4)

■ 가상환경 접속 및 해제와 삭제

- conda activate [선택할 가상환경 이름]
- conda deactivate
- conda env remove --name [삭제할 가상환경 이름]

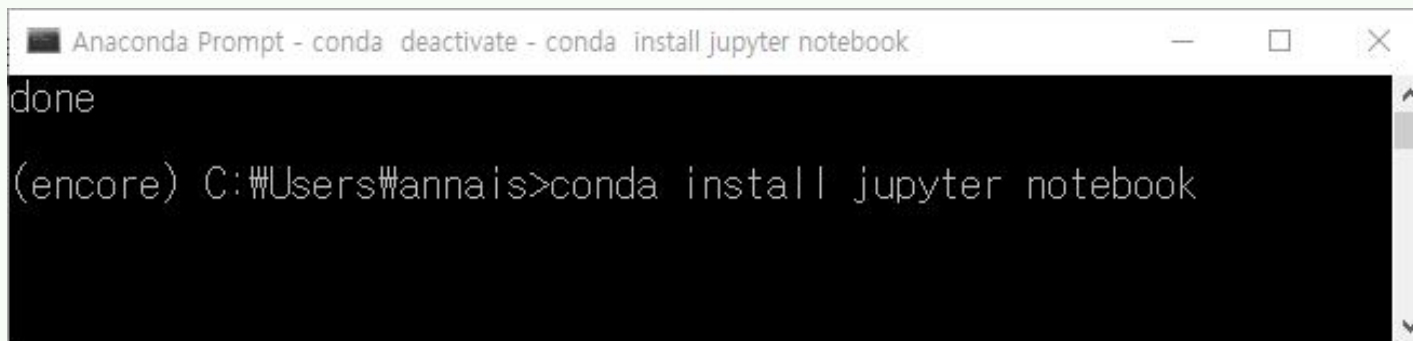


```
Anaconda Prompt - conda deactivate
tensorflow C:\Users\annais\Anaconda3\envs\tensorflow

(base) C:\Users\annais>conda activate encore
(encore) C:\Users\annais>conda deactivate
(base) C:\Users\annais>
```


가상환경 설정(5)

- Anaconda Navigator 또는 Anaconda Prompt 를 이용하여 가상환경을 설정 후, 생성한 가상환경으로 activate 한 다음 주피터 노트북을 설치함



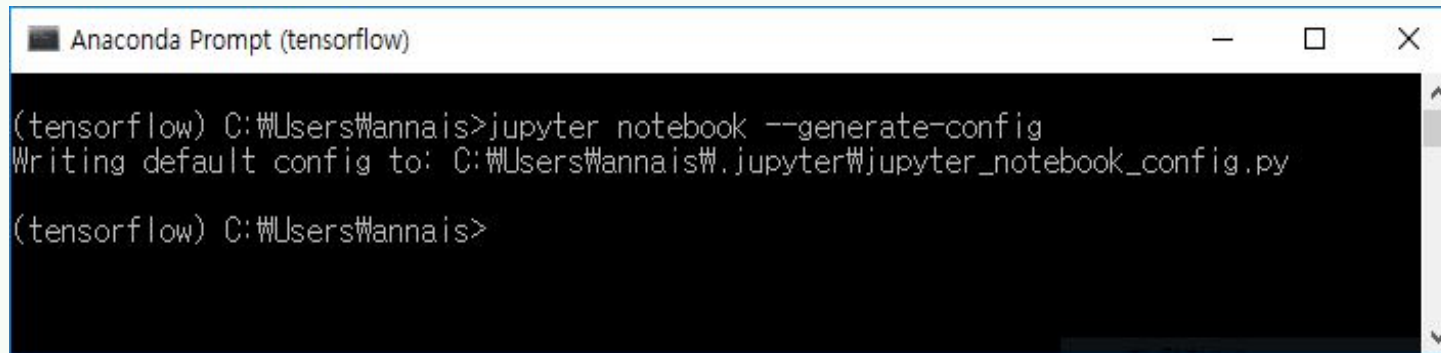
```
Anaconda Prompt - conda deactivate - conda install jupyter notebook
done
(encore) C:\Users\annais>conda install jupyter notebook
```

- 옵션으로
 - 주피터 노트북에서 python의 패키지를 관리할 수 있도록 해주는 명령어
 - 가상환경에서 필요한 패키지를 주피터 노트북의 Conda에서 관리 가능
 - `conda install nb_conda`

Jupyter notebook 설치

■ Jupyter notebook 실행 시 시작 위치 변경 방법

- Anaconda Prompt 열기
- 다음 명령을 통해 jupyter_notebook_config.py 생성하기
 - jupyter notebook --generate-config



```
Anaconda Prompt (tensorflow)
(tensorflow) C:\Users\Wannais>jupyter notebook --generate-config
Writing default config to: C:\Users\Wannais\jupyter\jupyter_notebook_config.py
(tensorflow) C:\Users\Wannais>
```

- 생성된 위치로 이동 후 문서편집기(예를 들어 노트패드 등)를 사용하여 파일 열기
- 문서찾기로 notebook_dir 찾기

```
## The directory to use for notebooks and kernels.
#c.NotebookApp.notebook_dir = "
```



```
## The directory to use for notebooks and kernels.
c.NotebookApp.notebook_dir = 'C:\User_folder'
```