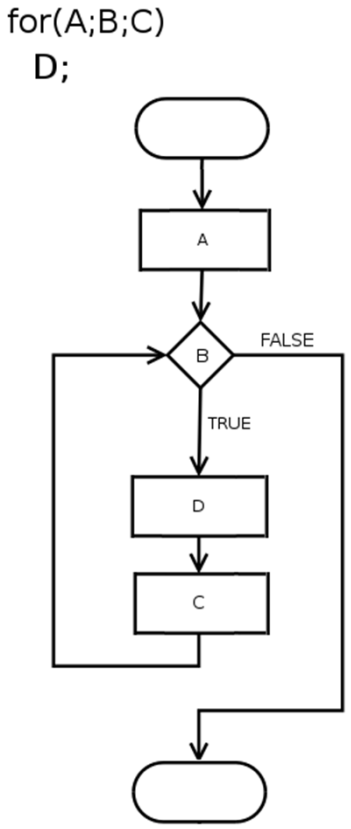
Clickseo.com

자료구조 & 알고리즘

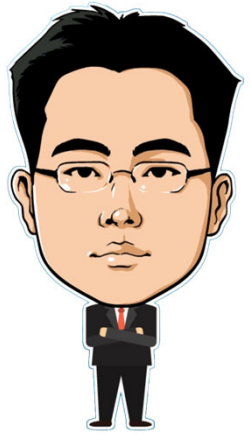


스택
(Stack)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



CC BY NC ND

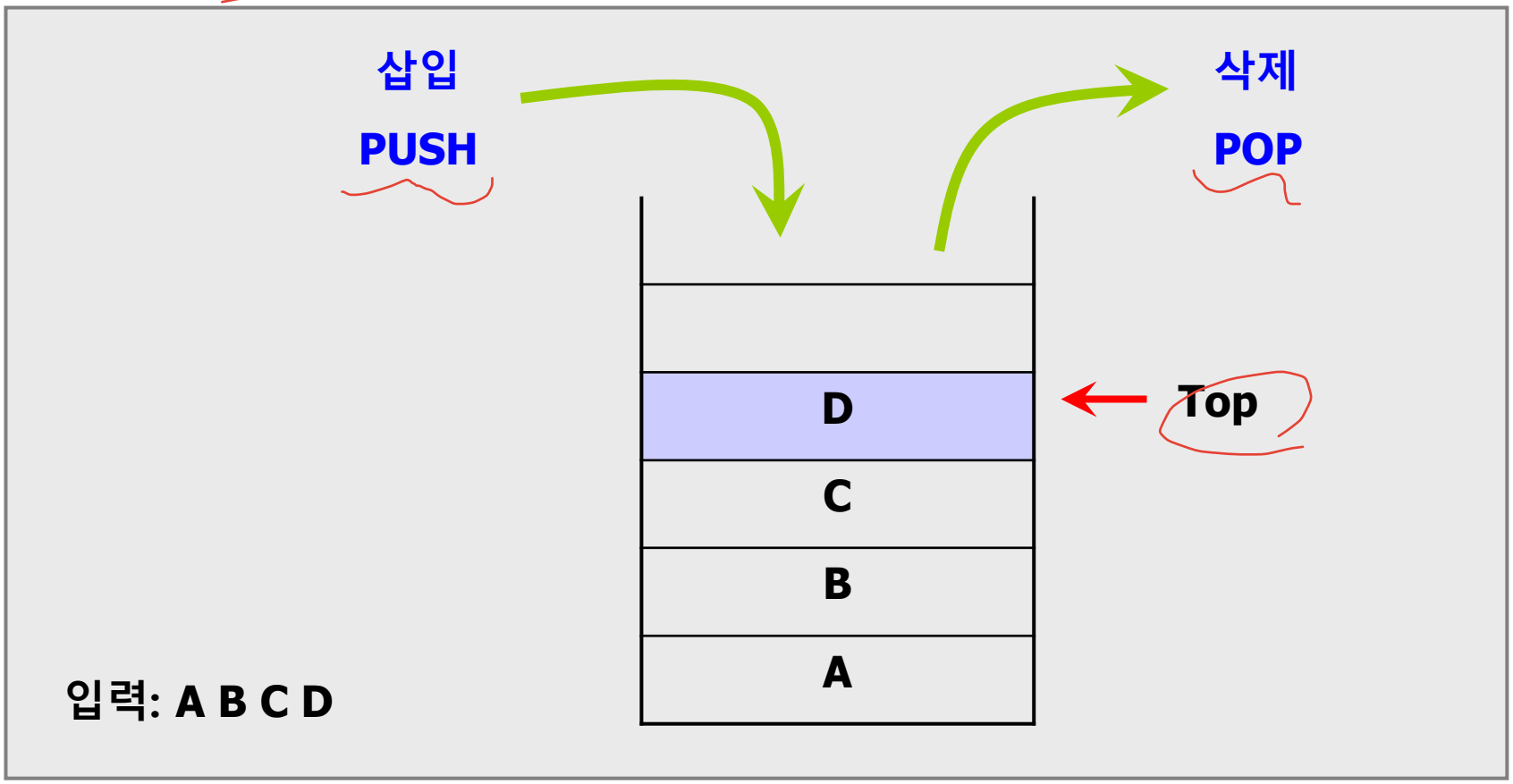
목 차

CC BY NC ND

- 스택의 이해
- 스택 구현
- 스택 응용

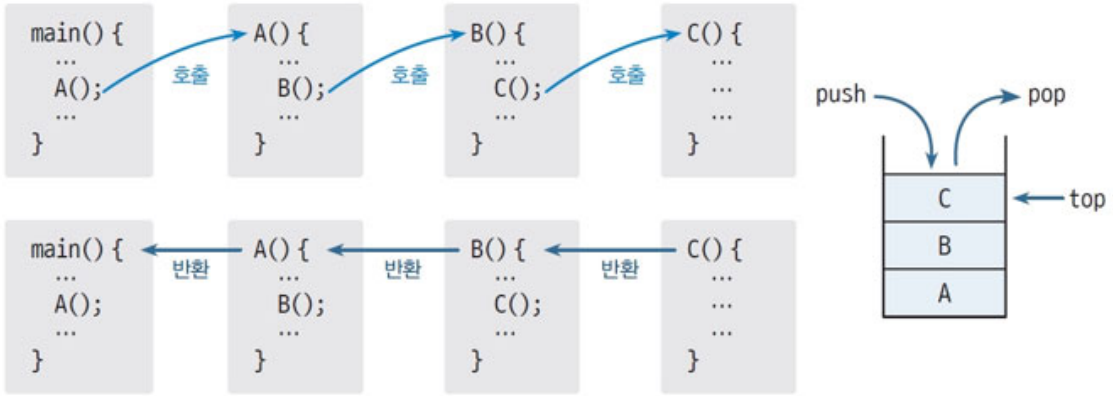
스택의 이해 (1/3)

- **스택**(Stack)
  - 후입선출(LIFO, Last-In-First-Out)



스택의 이해 (3/3)

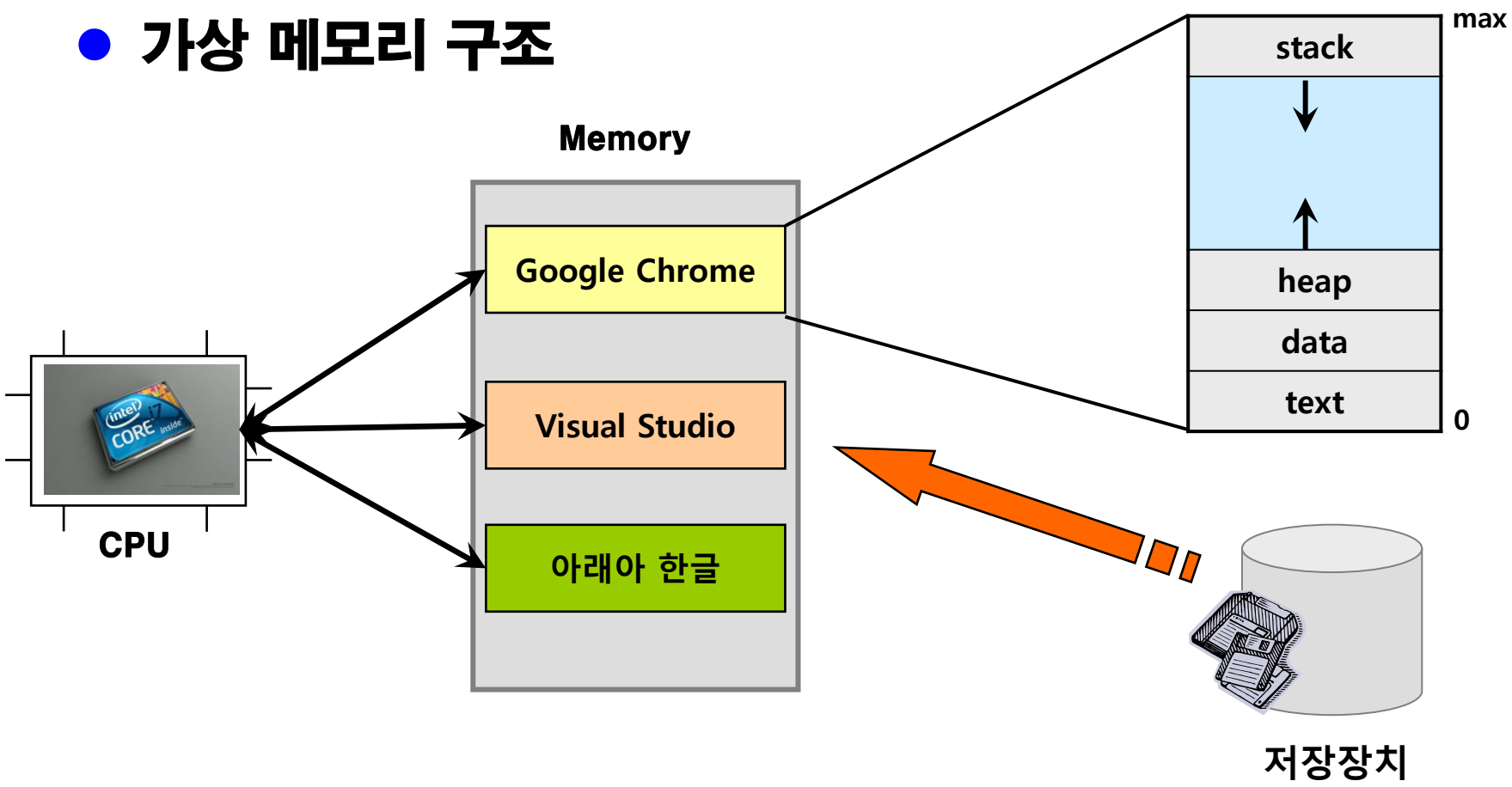
- **다양한 스택 활용**
  - **백 스페이스 키:** 최근에 입력한 글자 삭제
  - **최근에 작업한 순으로 취소: Ctrl + z**
  - **역순 문자열, 진법 변환**
  - **후위 표기법: 수식의 괄호 검사, 후위 표기법 변환과 수식 연산**
  - **시스템 스택**(System Stack)
    - 함수의 호출과 복귀 순서를 스택의 LIFO 구조를 응용하여 관리



[이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.]

스택의 이해 (3/3)

- 가상 메모리 구조



프로세스: 운영체제에서 프로세스는 "실행중인 프로그램"

프로그램: 컴퓨터를 실행시키기 위해 차례대로 작성된 "명령어 집합"

스택 구현

CC BY NC ND

- 스택의 이해
- **스택 구현**
  - **스택 구현:** 알고리즘
  - **스택 구현:** 순차.연결 자료구조
- 스택 응용

스택 구현: 알고리즘 (1/4)

- **스택 구현:** 알고리즘
  - 초기의 빈 스택 생성 알고리즘

```
stack_Create()
    stack[n];                    // 크기가 n 인 1차원 배열 생성
    top ← -1;
end stack_Create()
```

  - 스택에 최대로 저장할 수 있는 원소 개수를 배열 크기로 하여 1차원 배열을 선언
    - 저장된 원소가 없는 빈 스택 이므로 **top** 을 **-1**로 초기화

스택 구현: 알고리즘 (2/4)

- **스택 구현:** 알고리즘
  - 스택의 데이터 삽입 알고리즘: **PUSH**

```
push(S, data)
    if (top + 1 = n) then stack_isFull;
    else  S(++top) ← data;
end push()
```

  - 스택의 데이터 삭제 알고리즘: **POP**

```
pop(S)
    if (top = -1) then stack_isEmpty;
    else  return S(top--);
end pop()
```

스택 구현: 알고리즘 (3/4)

- **스택 구현:** 알고리즘
  - 스택의 공백 상태 검사 알고리즘

```
stack_isEmpty(S)
    if(top = -1) then return true;
    else return false;
end stack_isEmpty()
```

  - 스택의 포화 상태 검사 알고리즘

```
stack_isFull(S)
    if(top + 1 = n) then return true;
    else return false;
end stack_isFull()
```

스택 구현: 알고리즘 (4/4)

- **스택 구현:** 알고리즘
  - 스택 검색 알고리즘

```
Stack_Peek(S)
    if (top = -1) then stack_isEmpty;
    else  return S(top);
end Stack_Peek()
```

  - 스택에서 Stack[top]에 있는 원소를 검색하여 반환하는 연산

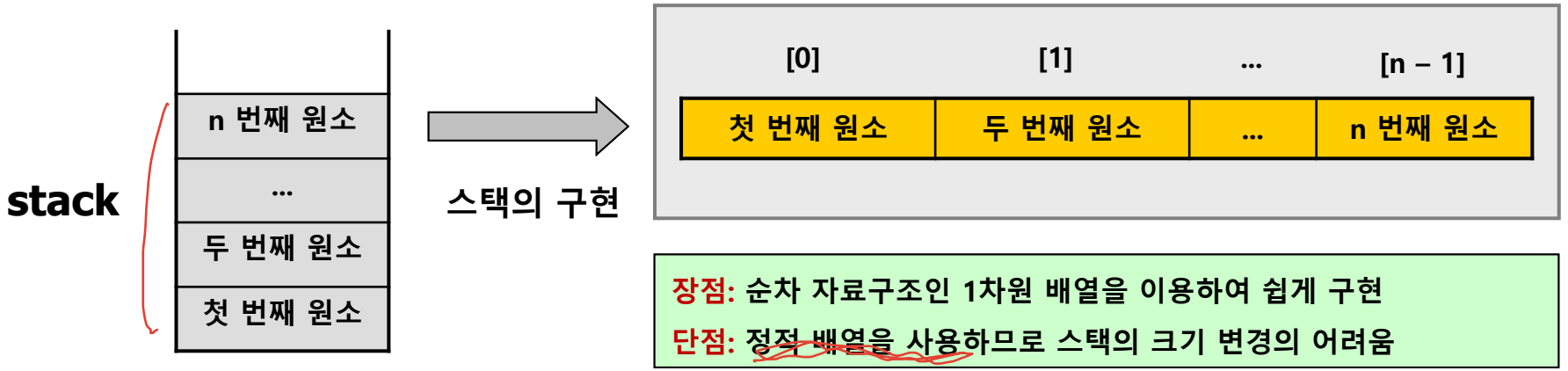
CC BY NC ND

스택 구현

순차.연결 자료구조

스택 구현 (1/2)

- **스택 구현:** 순차 자료구조
  - 순차 자료구조인 1차원 배열을 이용하여 구현
    - **스택 크기:** 배열의 크기
    - **스택에 저장된 원소의 순서:** 배열 원소의 첨자
    - 변수 top : 스택에 저장된 마지막 원소에 대한 첨자 저장
      - 공백 상태: top = -1 (초기값)
      - 포화 상태: top = n - 1

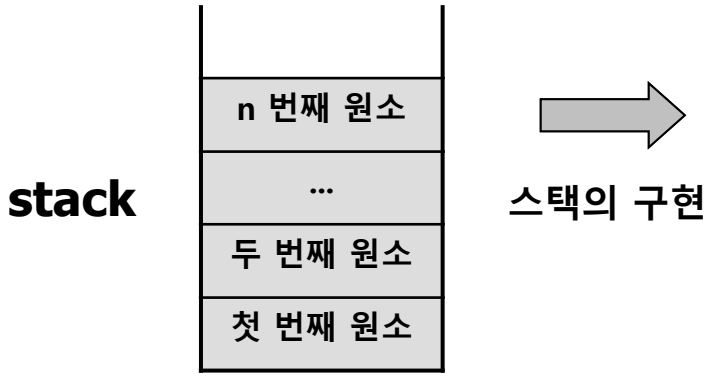


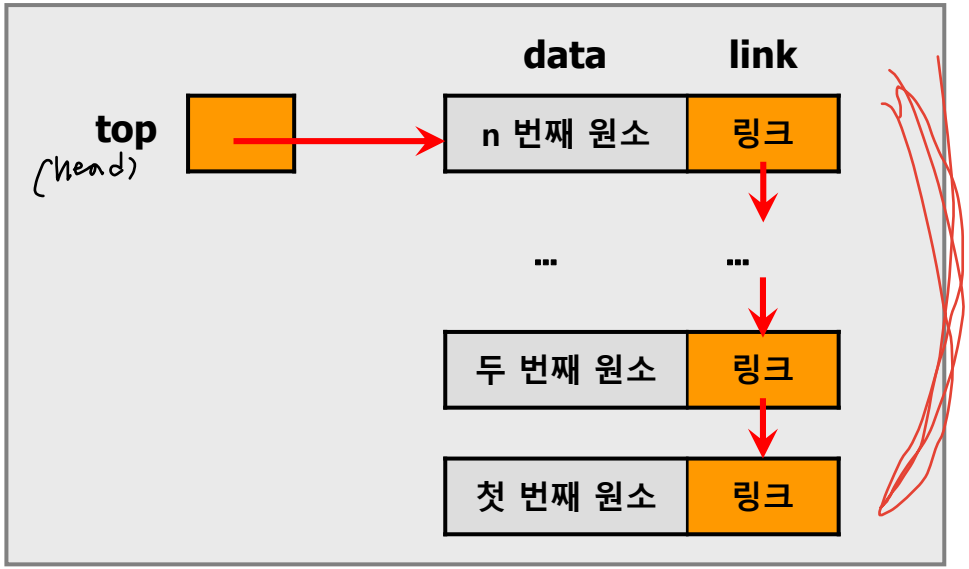
장점: 순차 자료구조인 1차원 배열을 이용하여 쉽게 구현

단점: 정적 배열을 사용하므로 스택의 크기 변경의 어려움

스택 구현 (2/2)

- **스택 구현:** 연결 자료구조
  - 단순 연결 리스트를 이용하여 구현
    - 스택의 원소: 단순 연결 리스트의 노드
      - **스택 원소의 순서:** 노드의 링크 필드로 연결
      - **push :** 항상 리스트의 첫 번째 노드로 삽입
      - **pop :** 항상 리스트의 마지막 노드를 삭제
    - 변수 top : 단순 연결 리스트의 마지막 노드를 가리키는 포인터 변수
      - 초기 상태: top = NULL





CC BY NC ND

스택 구현

순차 자료구조

스택 구현: 순차 자료 구조 (1/3)

- **스택 구현:** 순차 자료구조

C

```
typedef int element;
typedef struct _arrayStack {
    element         stack[stackMAXSIZE];
    int             top;
}arrayStack;

arrayStack *        stackCreate(void);
void                stackDestroy(arrayStack *);
void                push(arrayStack *, element);
element             pop(arrayStack *);
element             peek(arrayStack *);
_Bool               isEmpty(arrayStack *);
_Bool               isFull(arrayStack *);
void                printStack(arrayStack *);
```

스택 구현: 순차 자료 구조 (2/3)

- **스택 구현:** 순차 자료구조

```
template <typename E>
class arrayStack {
private:
    E                   stack[stackMAXSIZE];
    int                 top;
public:
    arrayStack();
    ~arrayStack();
    void    push(const E& e);
    E       pop(void);
    E       peek(void) const;
    bool    isEmpty(void) const;
    bool    isFull(void) const;
    void    printStack(void) const;
};
```

C++

스택 구현: 순차 자료 구조 (3/3)

- **스택 구현:** 순차 자료구조

```
class ListStack:
    def __init__(self):
        self.__stack = []
    def push(self, num):
    def pop(self):
    def peek(self):
    def isEmpty(self) -> bool:
    def popAll(self):
    def printStack(self):
```

Python

CC BY NC ND

스택 구현

연결 자료구조

스택 구현: 연결 자료 구조 (1/3)

- **스택 구현:** 연결 자료구조

C

```
typedef int element;
typedef struct _stackNode {
    element                 data;
    struct _stackNode*      link;
}stackNode;

typedef struct _ListStack {
    stackNode*      top;
}LinkedStack;

LinkedStack* stackCreate(void);
void         stackDestroy(LinkedStack *);
stackNode* makeStackNode(int num);
void         push(LinkedStack *, element);
element      pop(LinkedStack *);
element      peek(LinkedStack *);
_Bool        isEmpty(LinkedStack *);
// _Bool     isFull(LinkedStack*);
void                 printStack(LinkedStack *);
```

스택 구현: 연결 자료 구조 (2/3)

- **스택 구현:** 연결 자료구조

```
template <typename E>
class stackNode {
private:
    E                   data;
    stackNode<E>*       link;
    template <typename E> friend class LinkedStack;
};

template <typename E>
class LinkedStack {
private:
    stackNode<E>*       top;
public:
    LinkedStack();
    ~LinkedStack();
    stackNode<E>*       makeStackNode(const int& num) const;
    void                push(const E& e);
    E                   pop(void);
    E                   peek(void) const;
    bool                isEmpty(void) const;
    void                printStack(void) const;
};
```

C++

스택 구현: 연결 자료 구조 (3/3)

- **스택 구현:** 연결 자료구조

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

class LinkedStack :
    def __init__(self):
        self.__top = None
    def push(self, data) -> None:
    def pop(self):
    def peek(self):
    def isEmpty(self) -> bool:
    def printStack(self):
```

Python

스택 응용

cc BY NC ND

- 스택의 이해
- 스택 구현
- **스택 응용**
  - 후위 표기법

스택 응용 (1/3)

- **다양한 스택의 응용**
  - 역순 문자열
  - 백 스페이스 키
  - 진법의 변환
  - 수식의 괄호 검사
  - 수식의 후위 표기법 변환
  - 후위 표기법을 이용한 수식 연산
  - 시스템 스택(System Stack)
    - 함수의 호출과 복귀 순서를 스택의 LIFO 구조를 응용하여 관리

스택 응용 (1/2)

- **후위 표기법 변환:** 알고리즘

```
infix_to_postfix(exp)
    while (true) do
    {
        symbol ← getSymbol(exp);
        case
        {
            symbol = operand : print(symbol);
            symbol = operator :
                while (op(stack[top]) >= op(symbol)) do
                    print(pop(Stack));
                push(Stack, symbol);
            symbol = "(" : push(Stack, symbol);
            symbol = ")" :
                while (stack[top] ≠ "(" ) do
                    print(pop(Stack));
                pop(Stack);
            symbol = NULL : return;
        }
    }
end infix_to_postfix()
```

(3 * 5) – (6 / 2)

>> 3 5 * 6 2 / -

스택 응용 (2/2)

- **후위 표기법 연산:** 알고리즘

```
evalPostfix(exp)
    while (true) do
    {
        symbol ← getSymbol(exp);
        case
        {
            symbol = operand : push(Stack, symbol);
            symbol = operator :
                operand2 ← pop(Stack));
                operand1 ← pop(Stack));
                res ← operand1  op(symbol)  operand2;
                push(Stack, res);
            symbol = NULL : return;
        }
    }
end evalPostfix()
```

3 5 * 6 2 / - >> (결과) 12

참고문헌

[1] Michael T. Goodrich 외 2인 지음, 김유성 외 2인 옮김, “C++로 구현하는 자료구조와 알고리즘”, 한티에듀, 2020.

[2] “프로그래밍 대회 공략을 위한 알고리즘과 자료 구조 입문”, 와타노베 유타카 저, 윤인성 역, 인사이트, 2021.

[3] “IT CookBook, 쉽게 배우는 자료구조 with 파이썬”, 문병로, 한빛아카데미, 2022.

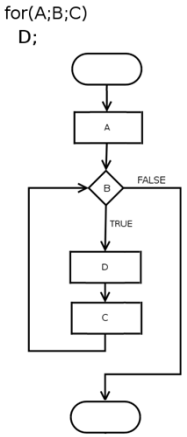
[4] “이것이 취업을 위한 코딩 테스트다 with 파이썬”, 나동빈, 한빛미디어, 2020.

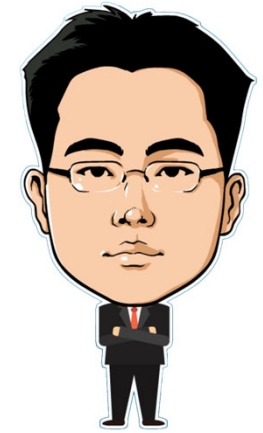
[5] 문병로, “IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법”(개정판), 개정판, 한빛아카데미, 2018.

[6] Richard E. Neapolitan, 도경구 역, “알고리즘 기초”, 도서출판 홍릉, 2017.

[7] 주우석, “IT CookBook, C・C++ 로 배우는 자료구조론”, 한빛아카데미, 2019.

[8] 이지영, “C 로 배우는 쉬운 자료구조”, 한빛아카데미, 2022.





이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.