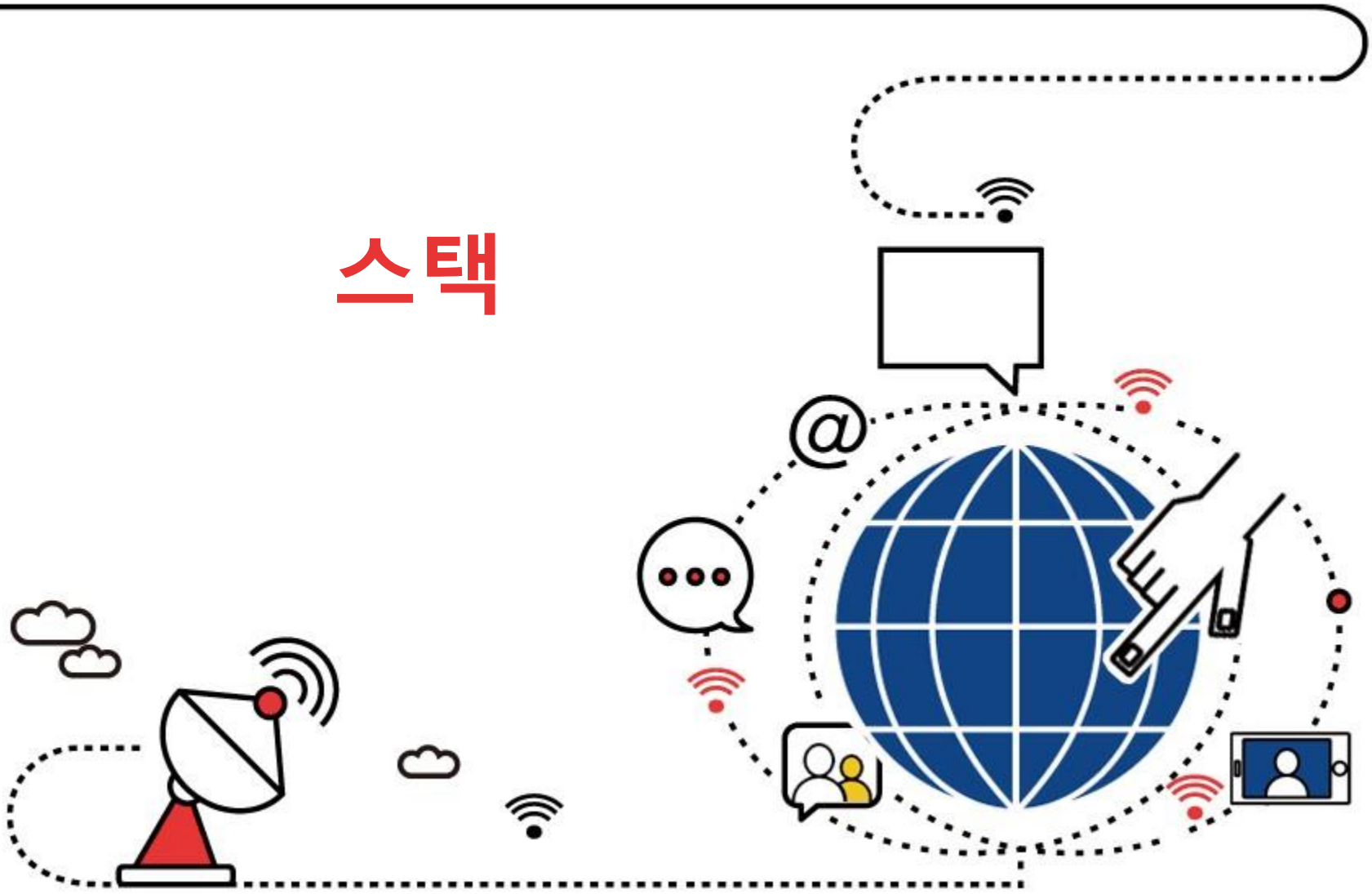
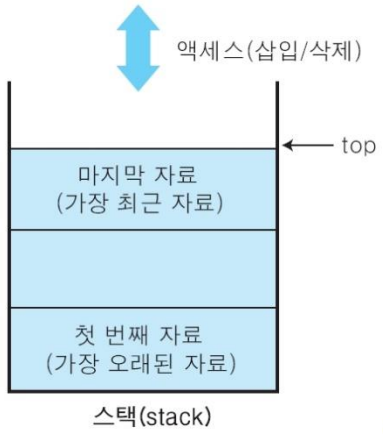
스택



1 스택

1. 접시를 쌓듯이 자료를 차곡차곡 쌓아 올린 형태의 자료구조 LIFO

2. 스택에 저장된 원소는 top으로 정한 곳에서만 접근 가능
   - top의 위치에서만 원소를 삽입하므로 먼저 삽입한 원소는 밑에 쌓이고, 나중에 삽입한 원소는 위에 쌓인다.
   - 마지막에 삽입(Last-In)한 원소는 맨 위에 쌓여 있다가 가장 먼저 삭제(First-Out)된다. → 후입선출 구조(LIFO, Last-In First-Out)

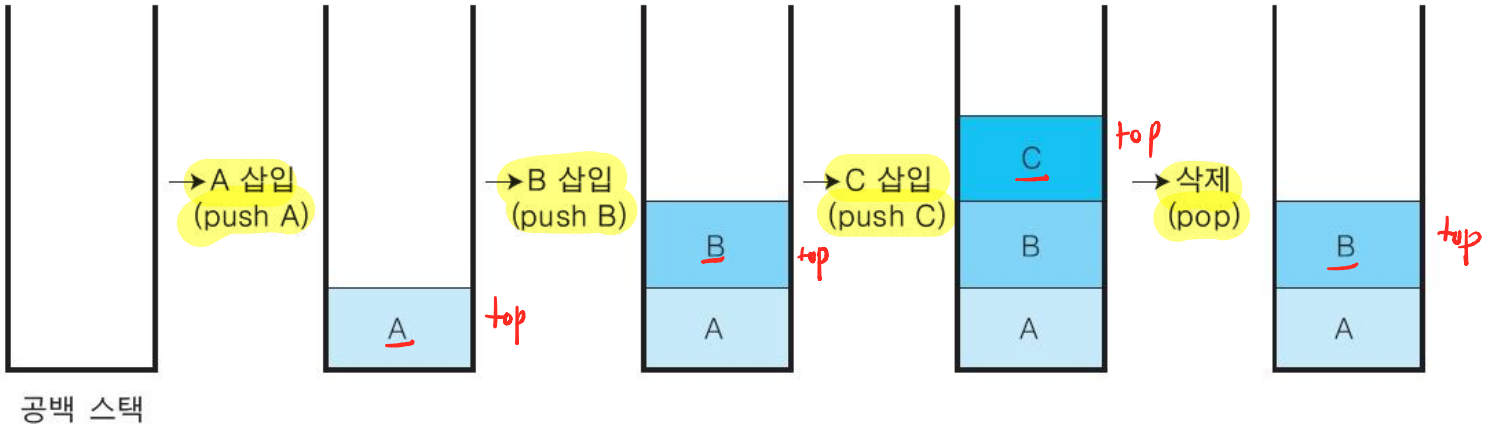


스택(stack)

2 스택의 연산

1 스택에서의 삽입과 삭제

- 공백 스택에 원소 A, B, C를 순서대로 삽입하고 한번 삭제하는 동안의 스택 변화



3 스택의 연산 알고리즘

1 스택의 push 알고리즘

① top ← top + 1;

- 스택 S에서 top이 마지막 자료를 가리키고 있으므로 그 위에 자료를 삽입하려면 먼저 top의 위치를 하나 증가

② S(top) ← x;

- 스택의 top이 가리키는 위치에 x 삽입

에러사항 : 꽉 찼다면 overflow → ∴ 배열 기반!

```
push(S, x)
        if (top == stack_size-1) then overflow;      // 초기상태: top=-1
        else {
                top ← top + 1; // ❶ 인덱스 1증가
                S(top) ← x;  // ❷ 원소 삽입!
        }
end push( )
```

3 스택의 연산 알고리즘

2 스택의 pop 알고리즘

① top ← top − 1;

- top의 위치는 그 아래 원소로 변경하기 위해 top 의 위치를 하나 감소

② return S(top+1);

- top이 위치를 변경하기 전의 top이 가리키는 원소를 반환

에러 처리 : 스택이 비어 있는 경우!

```
pop(S)
        if (top == -1) then error; // 스택에 남은 원소가 없으므로 pop 불가.
        else {
                top ← top – 1;        // ❶ 인덱스 1 감소
                return S(top+1); // ❷   pop한 값을 반환
        }
end pop( )
```

4 스택의 구현

1 배열을 이용한 구현

- 스택의 크기 : 배열의 크기
- 스택에 저장된 원소의 순서 : 배열 원소의 인덱스

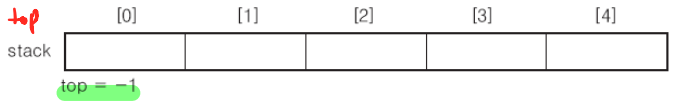
- 변수 top : 스택에 저장된 마지막 원소에 대한 인덱스 저장
  - 공백상태 : top = -1;
  - 포화상태 : top = n - 1;

4 스택의 구현

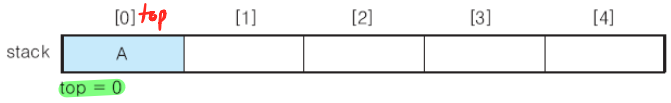
1 배열을 이용한 구현

- 크기가 5인 1차원 배열의 스택 연산 수행 과정

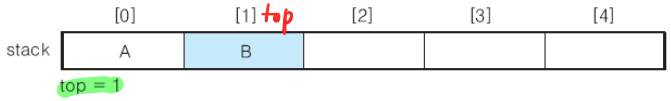
① 공백 스택 생성 : create(stack, 5);



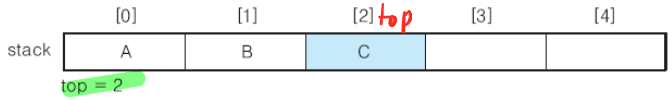
② 원소 A 삽입 : push(stack, A);



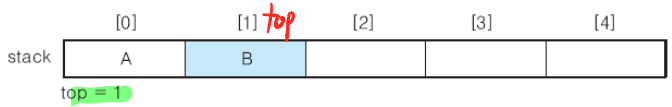
③ 원소 B 삽입 : push(stack, B);



④ 원소 C 삽입 : push(stack, C);



⑤ 원소 삭제 : pop(stack);



4 스택의 구현

1 배열을 이용한 구현

```
class arrayStack {
public:
    int* S;             // 배열
    int capacity;       // 배열의 크기  -> 이후 예시의 N
    int t;              // top의 index

    arrayStack(int capacity) {
        this->capacity = capacity;
        this->S = new int[capacity];   // 배열 S 할당
        this->t = -1;
    }
```

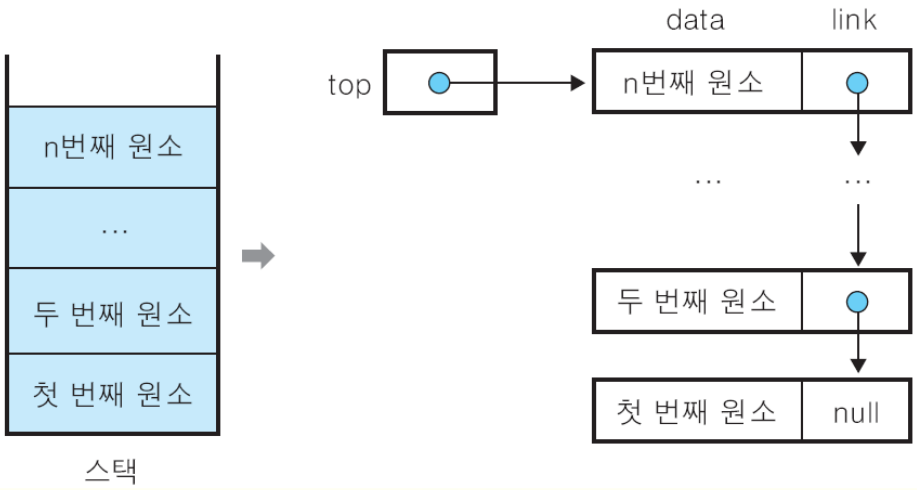
스택

```
    int size() {    // 스택의 원소 개수

    }
    bool empty() {   // 스택이 비었는지
        return
    }
    int top() {     // 스택의 맨 위 원소 확인
        if
        else
    }
    void push(int e) {  // 스택에 element 삽입


    }
    int pop() {    // 스택 element 삭제 & 반환
        if
        return
    }
```

4 스택의 구현

2 (단일) 링크드 리스트를 이용한 구현 Singly Linked List

- 스택의 원소 : 링크드 리스트의 노드
  - 스택 원소의 순서 : 노드의 링크 포인터로 연결
  - push : 리스트의 front(head)에 노드 삽입
  - pop : 리스트의 front(head) 노드 삭제
- 변수 Top : 단순 연결 리스트의 front 노드를 가리키는 포인터 변수
  - 초기 상태 : Top = null

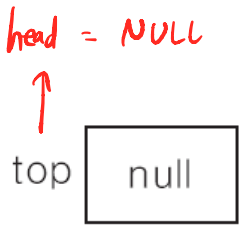


4 스택의 구현

2 링크드 리스트를 이용한 구현

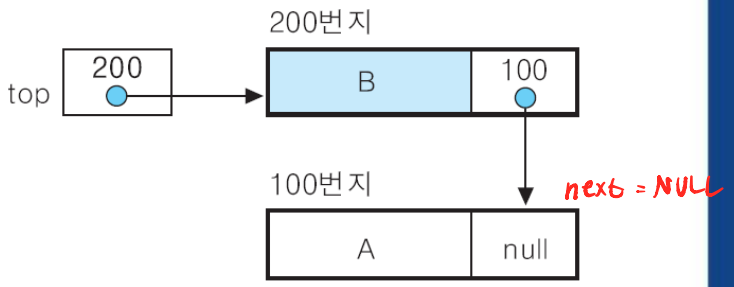
- 단순 연결 리스트의 스택 연산 수행 과정

① 공백 스택 생성 : create(S)



② 원소 A 삽입 : push(stack, A)

③ 원소 B 삽입 : push(stack, B)

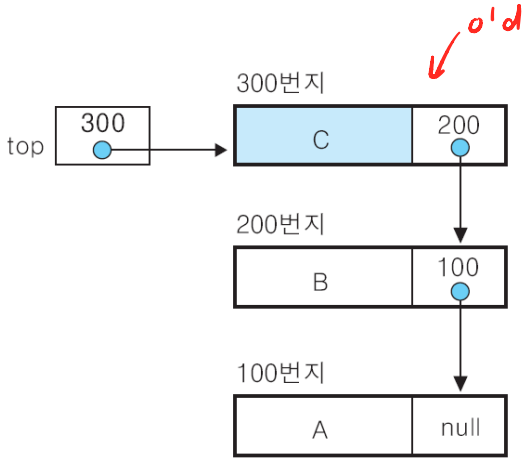


4 스택의 구현

2 링크드 리스트를 이용한 구현

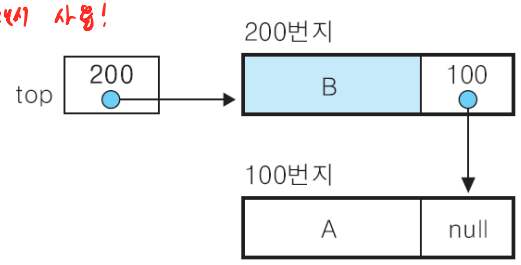
- 단순 연결 리스트의 스택 연산 수행 과정

④ 원소 C 삽입 : push(stack, C)



⑤ 원소 삭제 : pop(stack)

노드가 1개 이상 존재해 사용!



4 스택의 구현

2 링크드 리스트를 이용한 구현

```
class Node{
public:
    int data;
    Node* next;

    Node(int e) {
        this->data = e;
        this->next = NULL;
    }
};
```

```
class SlinkedList {
public:
    Node* head;
    Node* tail;

    SlinkedList() {
        head = NULL;
        tail = NULL;
    }
     int front() {
        return head->data;
     }
     void addFront(int e) {
        Node* newNode = new Node(e);
        if (head == NULL) {
            head = tail = newNode;
        }
        else {
            newNode->next = head;
            head = newNode;
        }
     }
     int removeFront() {
        Node* temp = head;
        head = head->next;
        return temp->data;
     }
```

스택

```
class linkedStack {
public:
    int n;    // 스택의 원소 개수
    SlinkedList* S;  // 링크드 리스트

    linkedStack() {
        this->S = new SlinkedList();
        this->n = 0;
    }
    int size() {  // 현재 스택의 원소개수 확인
        return
    }
    bool empty() {  // 스택이 비어있는지
        return
    }
    int top() {    // 스택의 맨 위 원소 확인
        if
        else
    }
```

```
void push(int e) {   // 스택 element 삽입


}
int pop() {   // 스택 element 삭제
    if
    else {


    }
}
```

5 스택의 응용

1 후위 표기식 연산

- **중위 표기법(Infix Notation)**
  - 연산자를 피연산자의 가운데 표기하는 방법

(피연산자)(연산자)(피연산자)

예) A+B A*B-C/D A-B*C+D

- **후위 표기법(Postfix Notation)**
  - 연산자를 피연산자 뒤에 표기하는 방법

(피연산자)(피연산자)(연산자)

예) AB+ AB*CD/- ABC*-D+

➡ 후위 표기법은 연산자 우선순위 따로 처리하지 않고 왼쪽에서 오른쪽으로 표기된 순서대로 처리할 수 있다

5 스택의 응용

1 후위 표기식 연산

중위 표기법 ➡ **후위 표기법** ➡ **후위 표기 수식연산**

- 후위 표기식 연산 방법
  - **예) AB*CD/-**

push(A)
push(B)
pop(B)
_B=B
pop(A)
_A=A
X=_A*_B
push(X)

push(C)
push(D)
pop(D)
_D=D
pop(C)
_C=C
Y=_C/_D
push(Y)

pop(Y)
_Y=Y
pop(X)
_X=X
Z=_X-_Y
push(Z)
pop(Z)

1, **피연산자**를 만나면 **스택**에 **push**한다.
2, **연산자**를 만나면 필요한 만큼의 **피연산자**를 **스택**에서 **pop**하여 연산 하고, 연산결과를 다시 **스택**에 **push** 한다.
3, 수식이 끝나면, 마지막으로 **스택**을 **pop**하여 출력한다.

5 스택의 응용

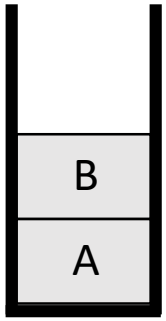
1 후위 표기식 연산

- **예) AB*CD/-**

① A B * C D / −

push

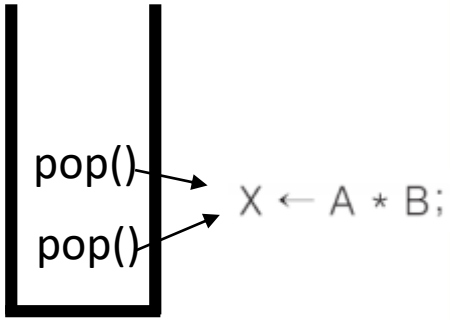
push



② A B * C D / −

pop()

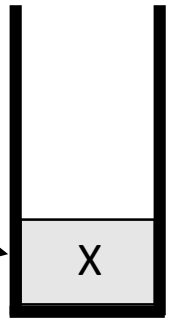
pop()



③

X

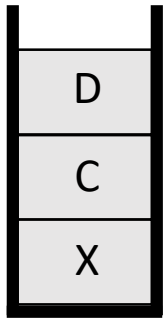
push



④ A B * C D / −

push

push

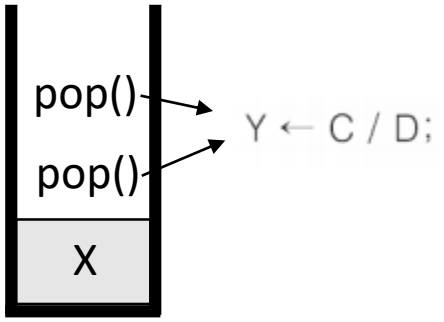


5 스택의 응용

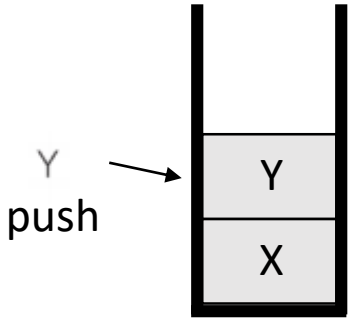
1 후위 표기식 연산

⑤ A B * C D / −

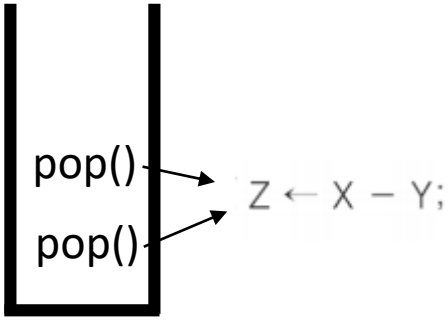
pop()
pop()



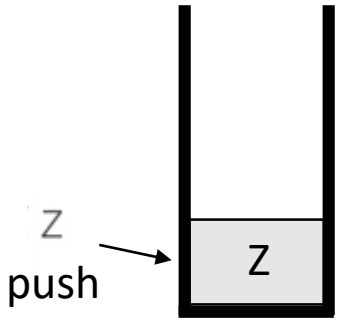
⑥



⑦ A B * C D / −



⑧



⑨

Z
pop()

```
int calPostfix(string exp) {
    Stack S = Stack();      // 스택
    string postfix = exp;   // 후위 표기식
    int n = postfix.size(); // 후위 표기식의 size
    char testch;  // 후위표기식 특정위치 문자 저장 변수

    for (int i = 0; i < n; i++) {
        testch = exp.at(i);
        if                                              { // 피연산자 처리

        }
        else { // 연산자 처리



            }
        }
    }
    return

}
```


Thank you