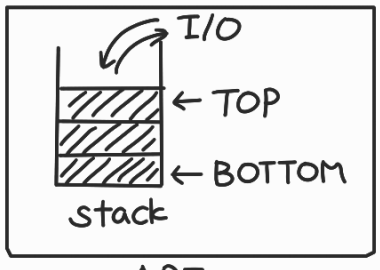
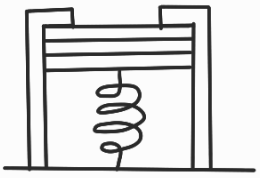
Ch 5 스택 (stack)

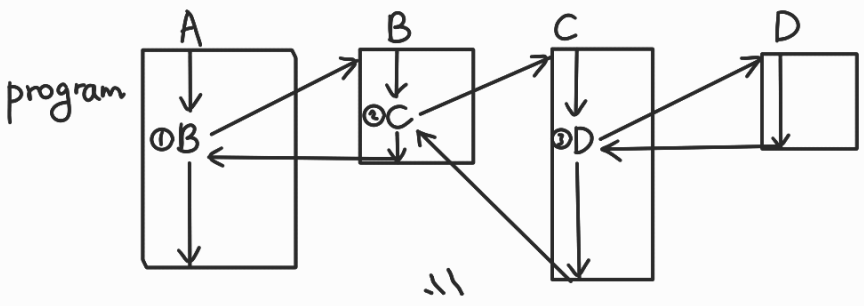
1. stack의 정의

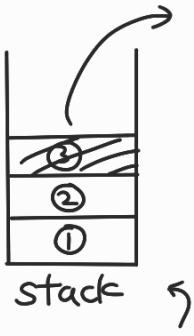


ADT ⇔

- top()
- push(e)
- pop()
- FULL()
- EMPTY()







Last - In First - out (LIFO)

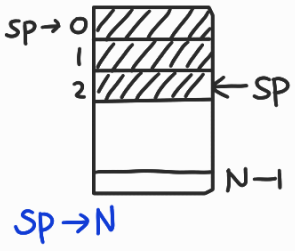
First - In First - out (FIFO)
↳ Queue

2. Stack의 구현

① 배열 (array)로 구현

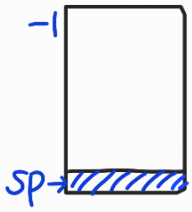
: 배열의 제한된 크기로 인한 stack overflow 고려

S[0 ... N-1]



sp (stack pointer) = 0 : 넣을 자리

push(e)	pop()
{ if(FULL) ret S[sp++] = e }	{ if(EMPTY) ret ret S[--sp] }
FULL : sp ≥ N	EMPTY : sp ≤ 0



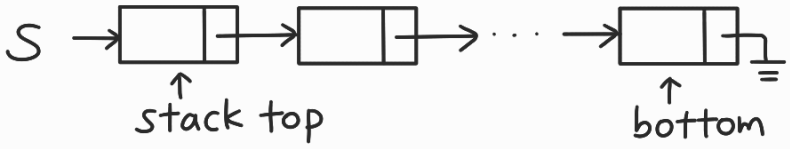
Sp = -1 : 넣은 자리

push(e)	pop()
{ if(FULL) ret S[++sp] = e }	{ if(EMPTY) ret ret S[sp--] }
FULL : sp ≥ N-1	EMPTY : sp < 0

② SLL (singly – linked list)로 구현

: 크기 제한이 없어 무한 삽입 가능

```
Node *S, *P
```



push(p) :  를 맨 앞에 삽입

```
{   p → Next = S
    S → p
}

Node *  pop()
{   if (S = NULL)  ret  NULL (empty)
    p = S
    S → S → Next
    ret p
}
```

3. stack의 이용 사례

① return address (함수호출 실행)

② Back Tracking (미로찾기, 그래프 탐색)

③ 식 (expression)의 처리 ↳ 중복X, 누락X

ⓐ = `b * c / (d + e)`

변수 = 식 : 할당문 (assignment statement)

식 expression	연산자 operator	피연산자 operand	결과	우선순위 priority
Arithmetic exp (산술식)	↑ * / + − 산술연산자	number	number	high ↓ low
Relational exp (관계식)	= ≠ > < ≥ ≤ 관계연산자		logical value	
Logical exp (논리식)	↑ NOT AND OR 논리연산자	logical value		

if (a == b || x > y+z) ...
② ③ ①

참고) Polish notation (식의 표기법)

a + b : infix notation (중위표기식) ① 변환
a b + : prefix " (전위 ") ② 계산
+ a b : postfix " (후위 ")

* 모든 식은 중위표기식 (infix) 에서 후위표기식 (postfix) 로 변환 후 계산

ex) a * (b + c) → a b c + *

a == b || x > y +

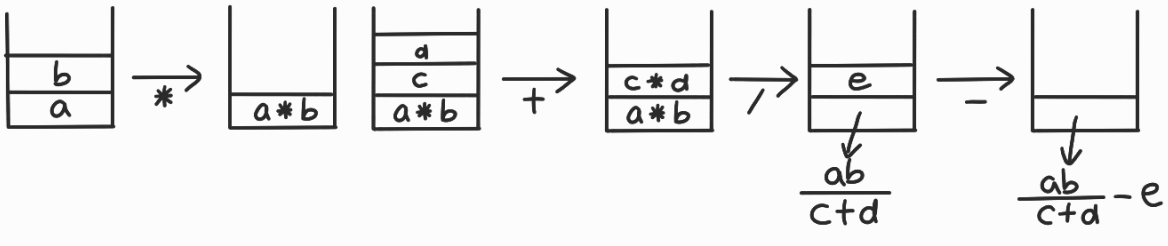
a == b || x > y + z → a b == x y z + > ||

ex) a * b / (c+d) - e
① ② ③ ④ : infix

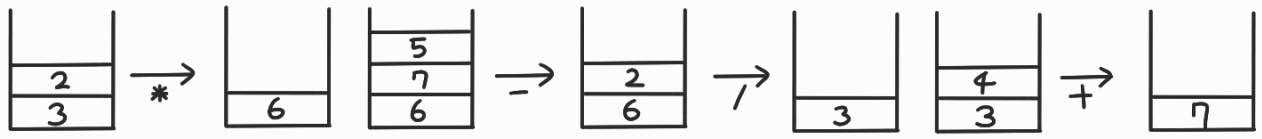
stack → − / * a b + c d e : prefix

stack → ab * cd + / e − : postfix

실제 계산은 stack 사용



ex) 3 2 * 7 5 - / 4 +



중위 → 후위 3+2+4*5+3/1

① 연산자 우선순위에 맞게 괄호를 쳐준다.

((3+(2+(4*5)))+(3/1))

② 괄호 안에 있는 연산자들을 뒤로 빼준다.

((3(2(45)*)+)+(31)/)+

③ 괄호를 모두 없애준다.

3245*++31/+

후위 표기법의 계산

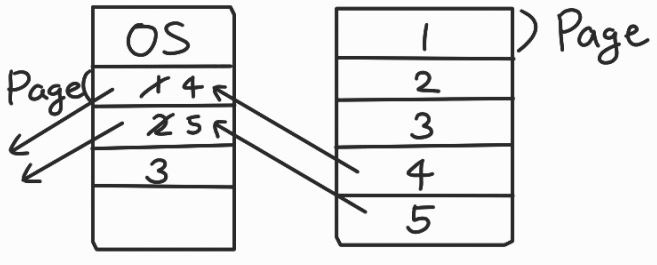
① 숫자가 나오면 전부 스택에 집어 넣는다.

② 연산자가 나오면 스택에서 두 수를 꺼내 계산하고 다시 스택에 집어넣는다.

④ LRU (Least - Recently Used) Stack

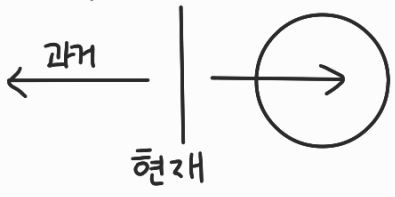
: 운영체제 (OS)가 program 실행 시 page 관리를 위해 사용하는 변형된 stack 으로서

- Demand Paging (요구 Paging) system 에서
- Page fault (page 부재) 가 발생하면
- Page Replacement (page 교체) 를 위해
- Victim Page (희생자 page)를 선정하는 방법



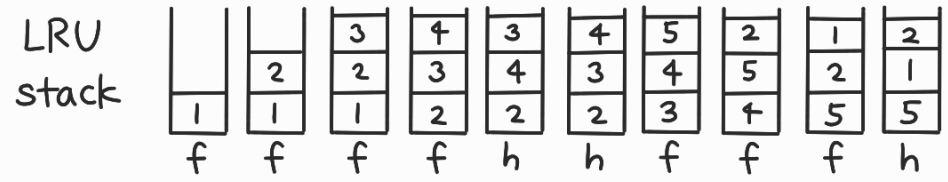
Hit-ratio → page fault rate의 반대 개념

LRU 정책



ex) 5-page program을 3-frame 공간에서 실행

호출 순서: 1 2 3 4 3 4 5 2 1 2



hit ratio = 0.3

fault rate = 0.6