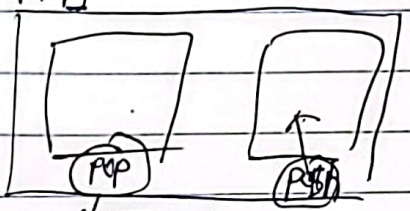
*: isFull() 체크해야할
pop() 꺼내서 X
peek() 꺼내서 확인

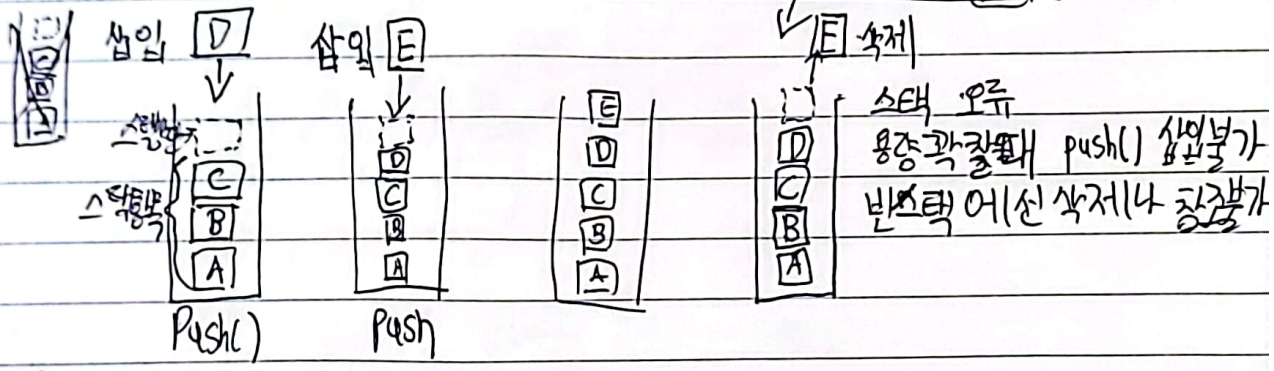
4. 스택

4.1 스택이란

데이터들 제한적으로 접근할수있는 구조
V 한쪽 끝에서만 자료를 넣거나 빼낼수있는 구조
가장 나중에 쌓은 데이터를 가장 먼저 빼낼수있는 데이터 구조
V 후입선출 (LIFO : Last-In First-Out)
V 가장 최근에 들어온 데이터 가장먼저나감



· 스택 (stack)
스택의 연산들



· 스택 ADT
데이터 : 후입선출 (LIFO)
stack() ; 비어있는 스택 만듦
isEmpty() ; 스택이 비어있을 시 True 아니면 False를 반환
Push() ; 항목 e를 스택의 맨위에 추가
pop() ; 스택의 맨위에 있는 항목을 꺼내 반환한다
peek() ; 스택의 맨위에 있는 항목을 삭제하지 않고 반환한다
size() ; 스택 내의 모든 항목들의 개수 반환
clear() ; 스택을 공백상태로 만든다

스택의 활용

웹페이지 뒤로가기

· 컴퓨터 내부의 프로세스 구현 등 동작방식

· 계산기

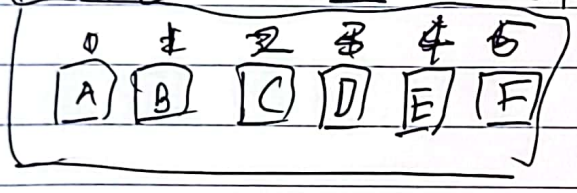
  v 후위 표기식 계산, 중위 표기식의 후위 표기식 변환

함수의 호출

· 미로 탐사

· 괄호검사

스택의 구현 배열구조



top : 스택 항목을 가정하는 파이썬 리스트 (인덱스)

항목의 개수는 len() 으로 구할수있음

스택의 구현 (배열구조)

바로 추가 가능 시간 O(1)

다른 항목의 이동이 필요 비효율적

· 스택의 구현 (배열구조) : 공백상태와 포화상태 검사

```
def isEmpty()
    if top == -1 : return True
    else : return False

def isFull
    return top == capacity-1
```

4.2 스택의 구현

새로운 요소 e를 삽입하는 : push(e)

```
def push(e) :
    global top
    if not isFull():
        top -= 1
        return array[top+1]
    else:
        print("stack underflow")
```

· 스택의 구현 (배열구조) ※코드 4.1 참조

상단요소를 저거하는 pop()

```
def pop():
    global top
    if not isEmpty():
        top -= 1
        return array[top+1]
    else:
        print("stack underflow")
        exit()
```

4.3 스택의 응용: 괄호 검사

괄호들이 같은 유형들끼리 쌍을 잘 이루어 사용되어야하는지 검사하는것

조건
1, 왼쪽괄호의 개수가 오른쪽 괄호의 개수와 같아야한다
2, 같은 타입의 괄호 가 오른쪽 괄호보다 먼저 나와야한다
3, 서로 다른 타입의 괄호 쌍이 서로를 교차하면 안된다

검사 방법

- 문자를 저장하는 스택을 준비한다, 처음에는 공백상태가되어야함
- 입력 문자열의 문자를 하나씩 읽어 왼쪽 괄호를 만나면 스택에 삽입한다
- 오른쪽 괄호를 만나면 pop연산으로 가장 최근에 삽입된 괄호를 꺼낸다
  이때 스택이 비었으면 조건 2에 위배된다
- 꺼낸 괄호가 오른쪽 괄호와 짝이 맞지 않으면 조건3에 위배된다
- 끝까지 처리했는데 스택에 괄호가 남아있으면 조건 1에 위배된다

괄호 검사 알고리즘

```
def checkBrackets(statement):
    stack = ArrayStack(100)
    for ch in statement:
        if ch == '{' or ch == '[' or ch == '(':
            stack.push(ch)
        elif ch == '}' or ch == ']' or ch == ')':
            if stack.isEmpty():
                return False
            else:
                left = stack.pop()
                if(ch == "}" and left != "{") or \
                (ch == "]" and left != "[" or \
                (ch == ")" and left != "(":
            return stack.isEmpty()
```

4.4 스택의 응용 : 수식의 계산

- 계산기 프로그램은 어떻게 만들까?

전위 (Prefix)	중위 (infix)	후위 (postfix)
연산자 피연산자1 피연산자2	피연산자1 연산자 피연산자2	피연산자1 피연산자2 연산자
+AB	A+B	AB+
+5*AB	5+A*B	5AB*+

중위표기 → 후위표기 → 후위표기 계산
변환

A * (B+C) / D - E

(((((A * (B+C)) / D) - E)

- 중위표기법 후위표기법 계산
- 2+3*4 → 234*+ → 14
- 모두 스택을 사용
- 먼저 후위 표기식 계산법을 알아보자

~~+BC*~~

~~ABC+*~~

~~ABC+*D/~~

~~ABCDE+*/-~~

~~BCADE+*/-~~

ABC+*D/E-

후위표기법 장점
- 괄호을 사용하지 않아도 계산순서를 알 수 있음
- 연산자의 우선순위를 생각할 필요가 없음
(식 자체에 우선 순위가 이미 포함됨)
- 수식을 읽으면서 바로 계산할 수 있음
(중위 표기법의 경우 수식을 끝까지 읽은 다음에 계산이 가능함)

(X+Y) - (W*Z)/V

~~XY+WZ*=~~

~~*+=~~

XY+WZ*V/-

2430

후위 표기 수식의 계산방법

82/3-32*+

82/3-32*+
43-32*+
132*+
16+7
7

4.13 중위식을 후위식으로 변환

5. A/B*C-D+E

[[{(A/B)*C}-D]+E]

AB/C*D-E+

A=6 B=3 C=2 D=5 E=4

알고리즘: 스택을 이용

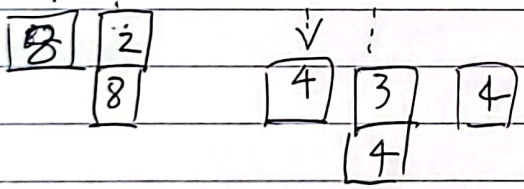
~~82/3-32*+~~

82/3-32*+

8/2=4

6/3*2-5+4=3

63/2*5-4+



63/2*5-4+

6 3 6 2 2 2 4 5 4 -1 4 -1 3
6 2 4 -1

- 알고리즘: 스택을 사용
1. 수식을 스캔하다가 피연산자가 나오면 스택에 저장
2. 연산자가 나오면 스택에서 피연산자 두개를 꺼내 연산을 실행하고 그 결과를 다시 스택에 저장
3. 이 과정을 수식이 모두 처리 될 때까지 반복
4. 마지막으로 스택에는 최종 계산 결과가 남음

안(OP) >= 밖(OP) : 안(OP).pop() / 출력(OP)
안(OP) < 밖(OP) : 밖(OP).push()

중위표기 수식의 후위 표기변환 // 피연산자 그냥출력
피연산자 순서가동일, '(' : 스택 밖: 연산자 우선순위 높음(가장)
- 중위와 후위표기법공통점 ~~후~~ 스택안 : 연산자 우선순위 낮음(가장)
- 연산자들의 순서만다름(우선순위 순서). ')' : 만날때까지 스택의 모든연산자들을 pop()
✓ 연산자만 스택에 저장후 출력
✓ 2+3*4 → 234*+

알고리즘
- 입력된 중위표기 수식은 순서대로 하나씩 스캔한다
- 피연산자를 만나면 문자를 그대로 출력
※ 연산자를 만나면 스택에 저장하되, 스택보다 우선순위가 낮은 연산자가 나오면 꺼내출력
- 왼쪽괄호는 우선순위가 가장낮은 연산자로 취급
- 오른쪽 괄호가 나오면 스택에서 왼쪽괄호위에 쌓여있는 모든연산자를 출력

시험 중위표기법 → 후위표기법 → 계산
스택 스택

8/2-3+(3*2)
82/3-32*+

```
def infix2Postix(expr):
    S = Stack()
    output = []
    for term in expr:
        if term in "(":
            S.push(()
        elif term in ')':
            while not S.isEmpty():
                op = S.pop()
                if op == '(': break;
                else:
                    output.append(op)
        elif term in "+-*/"
```

```
def precedence(op)
    if op == "(" or op = ")": return 0
    elif op == "+" or op == "-": return 1
    elif op == "*" or op == "/": return 2
    else: return -1
```

4.5 스택의 탐색) 스택의 응용, 미로탐색

미로탐색이란?

v 책 뒤 DFS, BFS와 유사
v 가장 간단한 방법
  v 시행착오
  v 하나의 경로를 선택하여 시도하다 막히면 다시 다른 경로를 선택하고
  v 현재의 경로가 막힐때 다시 선택할 수 있는 경로들을 어딘가에 저장 해야함
  (DFS, Dept First Search)

  벽: 1 길: 0 원지
  visited 한것인지 (방문)

탐색 알고리즘
- 시작위치를 스택에 넣는다
- 스택이 공백이 아니라면 하나의 위치를 꺼낸다. 이것이 현재위치이다 현위치에 방문했음 표시를 한다, 만약 스택이 공백일시 미로에 출구가 없으므로 종료
- 만약 현재위치가 출구라면 탐색 성공 종료
- 그렇지 않다면 이웃(상하좌우) 방들을 살핀, 만약 이웃 방들이 아직 방문 되지 않았고 갈수 있는 방이라면 그 방의 위치를 모두 스택삽입 반복

미로에 빠진 생쥐를 구출하자: DFS (Depth First Search)
-가던길이 막히면 가장 최근에 있었던 갈림길로 되돌아가서 다른 길을 찾는 방법 → stack 이용