Lifo   fifo
↑   ↑
Stack, Queue
Deque / 파이썬 고려함
→ 자료구조 기본위치

리스트는 임의의 위치에서도 삽입 [scribbled] 삭제 가능.

3. 리스트와 집합

↳ 항목들이 순서대로 나열되어 있고, 각 항목들은 위치를 갖는다

리스트 ⇒ 리스트 or 선형리스트

- 순서를 가진 항목들의 모임
- 데이터를 잘 관리하기 위해 묶어서 관리하는 자료형

집합 ⇒ 항목간의 순서개념이 없다

리스트 ADT

데이터 = 항목들이 순서대로 나열되어 있고, 각 항목들은 위치를 갖는다

연산

- insert(pos, e) pos 위치에 새로운 요소 e를 삽입한다
- delete(pos) pos 위치에 있는 요소를 꺼내고(삭제) 반환한다
- isEmpty() 리스트가 비어있는지를 검사한다
- GetEntry(pos) pos 위치에 있는 요소를 반환한다
- List() 비어있는 새로운 리스트를 만든다

리스트의 연산

- 삽입과 삭제: 리스트의 상태 변경 시킴
- 연산의 위치를 지정해야함
- 위치(인덱스)는 보통 0부터 시작
- 어떤 위치에 항목을 삽입/삭제하면 이후의 모든 자료들의 위치가 한칸씩 밀리거나 당겨짐 [A]

중간값 삭제 삽입하면 인덱스가 바뀐다.

[ ] ⇒ [A] ⇒ [A] [B] ⇒ [A] [C] [B] ⇒ [B] [C]

공백상태 리스트 → insert(0,A)

insert(1,B)

insert(1,C) → C를 삽입하면 이후의 B위치가 뒤로 밀림

Delete(0) → A(위치=0)를 삭제하면 이후의 모든(B,C) 항목이 앞으로 당겨짐

리스트구조성

복합자료형

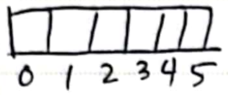
메모리 한정

구현방법

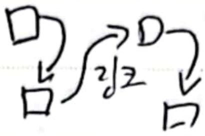
배열구조	연결된구조
• 구현이 간단 속도↑	• 삽입, 삭제효율 복잡함
• 항목접근이 O(1)	• 크기제한 X 속도↓

배열A

중간에 들어오면 인덱스값 바뀜



↗위치이동



중간에 들어와도 걱정X

배열구조	연결된구조
단점→삽입 삭제시 오버헤드	구현복잡
항목의 개수 제한	항목접근 O(n)

용어정리

리스트: 자료구조 리스트를 배열구조로 구현한 하나의 사례임

- 파이썬리스트: C언어에서의 배열이 강화된 형태의 스마트한 배열이다. 배열 또는 배열 구조의 의미로 사용
- 연결리스트: 자료들이 일렬로 나열할수 있는 연결된 구조를 말한다. 배열구조(파이썬리스트)에 대응되는 의미
- 자료구조리스트: 추상적인 의미의 자료구조 리스트를 의미한다.

구현→ 배열구조나 연결된구조 사용

추상적인 의미의 자료구조리스트, 배열 구조나 연결된 구조로 구현할수있다

자료구조리스트
- 파이썬의 list 클래스(배열구조)
- C++ STL의 vector 클래스(배열구조)
- C++ STL의 list 클래스(연결된구조)
- Java의 ArrayList 클래스(배열구조)
- Java의 LinkedList 클래스(연결된구조)

각 언어에서 자료구조 리스트로 구현한 사례들 파이썬리스트(list)는 배열구조로 구현되어있다

리스트 선언 A = [ ] 대괄호 (빈 공백) or A=list()

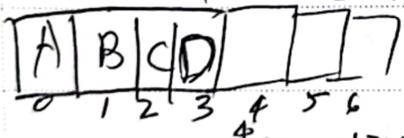
len(A)

=크기,항목수 출력

append → 항목 추가

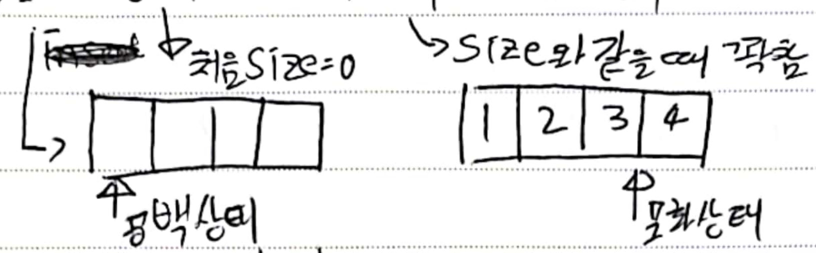
리스트
↳ 동적 배열로 구현 / 상황 크기 < 용량 , 크기 == 용량 일때
남은 공간이 없으면 어떻게 삽입 할까?
→ 기존의 메모리를 버리고 더 큰 새로운 메모리를 할당해 사용
- ○ 용량을 확장한 새로운 배열 할당
- ○ 기존의 배열을 새로운 배열에 복사
- ○ 항목을 삽입 → 기존 배열 해제, 리스트로 새 배열 사용

시간복잡도
append() 대부분의 경우 O(1)
insert() O(n) → 삽입할 자리 곳에 뒤쪽에 있는 항목들은 한칸씩 뒤로 이동해야함
pop() 연산 O(n) → 중간 원소를 삭제하면 그 뒤쪽 부분들은 한칸씩 앞으로 이동
리스트 ADT를 배열 구조로 구현
방법 1) 전역 변수와 함께 함수 이용 // 2) 클래스 이용
함수버전 ┌ 리스트 데이터를 전역변수로 선언
└ 리스트의 연산은 일반함수로 구현
훨씬 좋음 ↑
클래스버전 ┌ 리스트의 데이터를 클래스의 멤버변수로 구현
└ 리스트의 연산은 클래스의 메소드(멤버함수)로 구현
리스트구조 (배열)
좋은 방법은 아님 — 용량이 고정된 리스트구조



array[capacity] → 리스트 용량 -1
→ 항목들을 저장할 배열
size 전체항목수 -1
리스트의 데이터: 전역변수

isfull ↗ 포화상태와 공백상태 → is엠티



○ 삭제연산

A E B C D => A [ ] B C D 삭제한 항목을 복사해둠
0 1 2 3 4 delete(1)

E 뒤쪽 항목들은 앞으로 한칸씩이동

A B C D Size 감소, 복사한목 반환

5자리가 → 4자리로 줄어듬

○ 삽입연산

A B C D => A [ ] B C D 모든 항목들 뒤로 한칸씩이동
0 1 2 3 0 1 2 3 4

E 넣기? insert(1, E)

A E B C D 새로운 항목삽입
0 1 2 3 4

size를 증가 → 원래 4이었는 전에 지금은 5 만큼

○ 여러개의 리스트를 사용하려면?

○ 전역 변수와 함수로는 어려움

○ 클래스가 자료구조를 구현하는 가장 좋은 방법

함수 → 클래스 변경

○ 필요한 클래스를 선언한다

○ 전역변수로 선언된 데이터 => 클래스의 멤버변수 (생성자에서)

○ 일반함수로 구현된 연산 => 클래스의 멤버함수

○ 첫번째 매개변수로 self 추가 => 여기코드에서 그걸

○ 멤버함수에서 변수나 멤버함수를 호출하기위해 self. 사용

리스트의 응용 → 라인 편집기

기능 => i 라인 삽입 p 현재내용출력

d 라인삭제 l 파일입력

r 라인 변경 s 파일 출력

집합 => $S = \{item_0, item_1, item_2 \cdots item_{n-1}\}$

특성보기
- 원소의 중복을 허용하지않음
- 원소들 사이에 순서가 없음 (선형자료구조가 아님)

집합ADT

데이터 : 같은 유형의 구별되는 요소들의 모임, 원소들은 순서는 없지만 서로 비교할 수는 있어야함

연산

- set()
- size()
- contains(e)
- insert()
- delete()
- equals
- union 합집합
- intersect 교집합
- difference 차집합
- display

구현방법 → 다양한 방법으로 구현가능
- 배열, 비트벡터, 트리, 해싱 구조등
- 사용 방법에 따라 연산들의 성능이 달라짐

집합연산들
- 원소 e가 집합에 포함되는지 contains(e) 연산 → 순차탐색, O(n)
- 원소를 삽입하는 insert(e)연산 → 중복검사, O(n) // 맨뒤에 저장
- 원소를 삭제하는 delete 연산
  - 삭제할 원소를 찾기, O(n) → 있으면 그 원소삭제 => 이후의 모든 원소 이동 →
  - 개선 : 맨 뒤의 원소를 삭제할 원소위치로 옮김
- 기타연산
  - 포함
  - 합집합
  - 차집합

→ 구현 방법에 따라 연산들 성능 영향받음